VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INTELIGENTNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INTELLIGENT SYSTEMS

VIZUALIZACE PLÁNOVÁNÍ CESTY PRO NEHOLONOMNÍ OBJEKTY

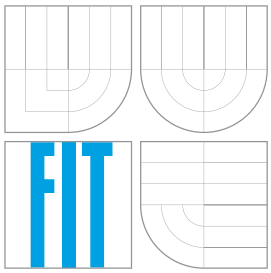
DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

Bc. JAN OHNHEISER

BRNO 2013

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INTELIGENTNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INTELLIGENT SYSTEMS

VIZUALIZACE PLÁNOVÁNÍ CESTY PRO NEHOLONOMNÍ OBJEKTY

VISUALISATION OF PATH-PLANNING FOR NONHOLONOMIC OBJECTS

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

Bc. JAN OHNHEISER

VEDOUCÍ PRÁCE
SUPERVISOR

Ing. JAROSLAV ROZMAN, Ph.D.

BRNO 2013

Abstrakt

Tato práce se zabývá plánováním cesty neholonomního robota použitím pravděpodobnostních algoritmů. V teoretické části je rozebrán obecně problém hledání cest. Následně se pak práce zaměří na pravděpodobnostní algoritmy. Praktická část se zabývá návrhem a implementací appletu a webových stránek, jenž demonstrují některé druhy pravděpodobnostních algoritmů na uživatelem zadaných objektech.

Abstract

This work deals with the path finding for nonholonomic robots using probabilistic algorithms. The theoretical part analyzes the general problem of finding routes. Subsequently, the work will focus on probabilistic algorithms. The practical part describes design and implementation of the applet and web sites that demonstrate probabilistic algorithms to user-specified objects.

Klíčová slova

Pravděpodobnostní algoritmy, hledání cesty, robotika, neholonomní robot, PRM, RRT, EST, SRT.

Keywords

Probabilistic algorithm, patch finding, robotics, nonholonomic robot, PRM, RRT, EST, SRT.

Citace

Jan Ohnheiser: Vizualizace plánování cesty pro neholonomní objekty, diplomová práce, Brno, FIT VUT v Brně, 2013

Vizualizace plánování cesty pro neholonomní objekty

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením pana Ing. Jaroslava Rozmana Ph.D.

.....
Jan Ohnheiser
22. 5. 2013

Poděkování

Rád bych poděkoval svému vedoucímu Ing. Jaroslavu Rozmananovi, Ph.D. za ochotu a čas strávený při konzultacích této práce. Také děkuji své rodině za stálou podporu při studiu.

Obsah

Seznam obrázků

Kapitola 1

Úvod

Robotika je populární obor, který vznikl ve dvacátém století. Slovo „robotika“ poprvé použil Isaac Asimov roku 1941 v povídce „Runaround“. Během času se ze science fiction stala realita. Robotika se zabývá návrhem a konstrukcí robotů, které pomáhají nebo úplně nahrazují lidskou práci. Nejprve se objevili průmysloví roboti s jednoduchou logikou, kteří měli za úkol periodicky opakovat zadaný pohyb, čímž umožnili zefektivnění sériové výroby.

V dnešní době obor robotika zažívá velký rozvoj. Za posledních několik let vzniklo nespočet nových technologií. Využití robotiky je nesmírně rozsáhlé a má velký potenciál. Dnes plně samostatné stroje prozkoumávají cizí planety, nebo provádějí chirurgické operace, zachraňují člověka v hořícím či zříceném domě.

Tato diplomová práce se zabývá problematikou hledání cest spadající do oboru robotiky, avšak využití najdeme i v jiných oborech. Cílem hledání cesty je, jak už z názvu vyplývá, najít cestu z počáteční konfigurace systému do cílové konfigurace. Rychlé a efektivní řešení může zachránit životy.

Tato práce se zaměřuje na hledání cest za použití pravděpodobnostních algoritmů. Jedná se o prohledávání spojitého prostoru. Možné je použití i ve více dimenzionálních prostorech a u robotů s více stupni volnosti. Základní myšlenkou pravděpodobnostních algoritmů je vzorkování volného konfiguračního prostoru a následného vytvoření grafu, nad kterým probíhá prohledávání (například pomocí Dijkstrova algoritmu).

Pravděpodobnostní algoritmy jsou pravděpodobnostně úplné, to znamená, že pokud existuje cesta, algoritmus má určitou nenulovou pravděpodobnost jejího nalezení. Doba výpočtu tedy zvyšuje pravděpodobnost nalezení nějakého řešení.

Dalším přitížením pro hledání cesty je problém plánování cesty „neholonomního“ robota. Jde totiž o roboty, kteří se nemohou pohybovat všemi směry nezávisle na momentální konfiguraci nebo nemohou libovolně měnit svou orientaci. Při pohybu otáčejí koly nebo jinými prostředky, které jim umožní měnit trajektorii pohybu.

Využití plánování cest pro neholonomní roboty je velmi rozsáhlé. Díky němu můžeme nechat samo podélně zaparkovat auto, nebo auto vyrobit pomocí robotického ramena, nebo dokonce dokovat raketoplán k vesmírné stanici.

1.1 Cíle práce

Cílem této diplomového práce je nastudování problematiky hledání cest pomocí pravděpodobnostních algoritmů, dále pak zaměření na problematiku hledání cest pro neholonomní objekty. Výstupem práce má být ukázkový applet a webové stránky s teoretickým popisem problému.

Applet bude celý problém hledání cest názorně vizualizovat s možností výběru několika různých pravděpodobnostních metod hledání cest s různými parametry, tvary a vlastnostmi robota. Applet umožní měnit prostředí, ve kterém se robot pohybuje, a jednotlivé algoritmy krokovat, případně výpočet zrychlovat nebo zpomalovat.

1.2 Skladba práce

První část popisuje teorii, která se práce týká, jsou to kapitoly 2 a 3. Zbývající kapitoly se zabývají appletem, jeho tvorbou a zhodnocením.

Kapitola 2 Robot a okolní prostředí popisuje, co přesně znamená neholonomní robot a jaké existují základní typy neholonomních robotů, dále také různé reprezentace prostředí a interakce mezi robotem a jeho okolím. Kapitola 3 Plánování cesty popisuje metody plánování cest a speciálně se zaměřuje na pravděpodobnostní metody a techniky používané pro neholonomní objekty.

Kapitola 4 Návrh appletu a webových stránek obsahuje zvolené techniky a navržený diagram tříd, podle kterého byl applet implementován. Kapitola 5 Implementace pak popisuje postup při implementaci a konkrétní řešení jednotlivých problémů. Kapitola 6 Popis appletu obsahuje návod k ovládání appletu, popis uživatelského rozhraní a chování appletu. Kapitola 7 Testování obsahuje sbírku testů, kterými applet prošel, a získané výsledky. Závěrečná kapitola zhodnocuje applet a jeho výsledky získané testováním.

Kapitola 2

Robot a okolní prostředí

Poprvé se objevilo slovo robot ve hře Karla Čapka RUR. Slovo robot je tak dnes nejznámějším českým slovem na světě. Robot je mechanické zařízení, které slouží pro usnadnění či nahrazení lidské práce. Robot může pracovat pod přímým řízením člověka nebo autonomně pod řízením naprogramovaného počítače.

2.1 Autonomní mobilní robot

Pojem autonomní mobilní robot označuje robota, který má možnost pohybu a dokáže řešit určitý problém nebo i více problémů samostatně bez zásahu člověka. O těchto Robotech lze napsat mnoho, jak z fyzické složky, jeho mechanických možnostech, tvaru a dalších vlastností, tak i té inteligentní složky, uvědomování si okolního prostředí a plánování kroků nutných k dosažení cíle. Tato kapitola je věnována vlastnostem těchto robotů. (Následující informace byly čerpány z opory Robotika [12])

Robot musí znát své fyzikální vlastnosti. Musí znát svou velikost a odhadovat, kam se vejde či nikoliv. Musí vnímat okolní prostředí, aby mohl reagovat na různé podněty. A především musí umět i prostředí ovlivňovat. Autonomní mobilní robot by měl mít tyto základní subsystémy:

- Senzorický subsystém – schopnost vnímat prostředí – senzory, snímače,
- Motorický subsystém – schopnost ovlivňovat prostředí – efektory,
- Kognitivní subsystém – schopnost myslet, plánovánovat a tvořit model prostředí.

2.2 Neholonomní robot

Roboty můžeme podle možností pohybu dělit na *holonomní* a *neholonomní*. *Holonomní robot* je takový, který se může pohybovat ve všech směrech a libovolně se při tom otáčet. Jeho počet stupňů volnosti je roven celkovému počtu stupňů volnosti v daném prostředí. Tito roboti jsou obecně obtížně realizovatelní počítáme-li s dvourozměrným prostředím, natož pak ve třech rozměrech. Zajímavý popis této problematiky je k nalezení na [1].

Neholonomní robot naopak nemá možnost libovolně zrychlovat do všech směrů či otáčet se libovolně na místě. Jeho počet stupňů volnosti je menší než celkový počet stupňů volnosti v daném prostředí. Příkladem neholonomním objektům mohou být téměř všechny dopravní

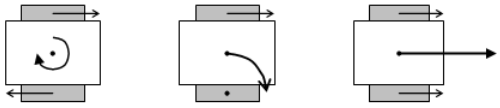
prostředky, které dnes používáme. Patří sem kolové a pásové vozidla, letadla a lodě, ale i vesmírná plavidla.

Uvažujme projekci do jednorozměrného prostoru. Jako holonomní stroj pak můžeme považovat například vlak na kolejích. V případě dvourozměrné projekce pak existuje robot se speciálním podvozkem, jinak nazývaný jako všesměrový. Robot je vybaven koly, jenž jsou schopny pohybu do všech směrů, nebo otáčivých kol alespoň do úhlu 180° (se zanedbáním kol samotných), robot je tak schopen otáčet a pohybovat se jakkoliv potřebuje.

Tato práce se však zabývá neholonomními objekty, práce s nimi je ale mnohem složitější.

2.2.1 Diferenciální podvozek

Diferenciální typ podvozku je jedním z nejpoužívanějších typů kolových podvozků u robotů. Zvláště pro jeho jednoduchost si jej oblíbili amatérští konstruktéři robotů. [3]



Obrázek 2.1: Příklad neholonomního objektu s diferenciálním podvozkem (tank)

Podvozek se skládá ze dvou hnacích kol nebo pásů, popřípadě jako opěrný bod pro stabilitu otočné kolečko (jako nákupní vozík). Obě hnací kola jsou řiditelné zvlášť, nejčastěji stejnosměrnými nebo krokovými elektromotory. Ty pak zajišťují schopnost pohybu a zároveň otáčení na místě (viz obrázek 2.1). Při pohybu vpřed jsou oba motory poháněny stejnou rychlostí. Při otáčivém pohybu záleží na velikosti rozdílu mezi rychlostmi obou motorů (proto diferenciální). Rychlost pohybu vpřed je dána jako průměr rychlostí obou kol. Úhlová rychlost otáčení je pak dána rozdílem rychlostí obou kol.

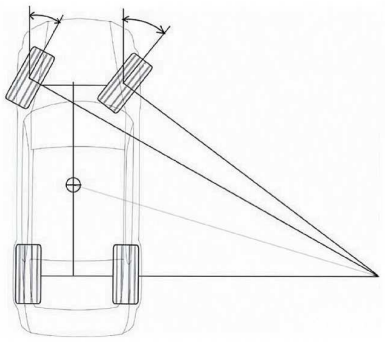
Konfiguraci robotu ve zjednodušené dvourozměrné projekci prostoru lze vyjádřit jako trojici $x = (x_1, x_2, x_3)$, kde x_1 a x_2 určují pozici auta v prostoru a x_3 úhel natočení auta. Nechť kola jedou rychlostí v_1 a v_2 a jsou vzdáleny od středu robota na vzdálenost l, pak můžeme psát diferenciální rovnici pro vektor konfigurace 2.1.

$$x' = v_1(\frac{1}{2}cos(x_3), \frac{1}{2}sin(x_3), \frac{1}{l}) + v_2(\frac{1}{2}cos(x_3), \frac{1}{2}sin(x_3), \frac{-1}{l}) \tag{2.1}$$

2.2.2 Ackermanův podvozek

Ackermanovo řízení je dnes nejvíce rozšířené v pozemní dopravě v podobě dnešních aut. Podvozek se skládá ze tří a více kol, ty dále rozdělíme na otočná kola a pevná. Neotočná kola sdílejí jednu osu, otáčivá kola jsou při změně směru otočena tak, aby průsečíky mezi všemi osami kol tvořily společný jeden bod (nejsou-li momentálně v rovnoběžné pozici jízdou rovně), tento průnik je pak středem otáčení celého vozidla. Při výpočtech se předpokládá, že auto při jízdě nepůjde do smyku. [3]

Z příkladu robota na obrázku 2.2 je jasné, že se jedná o neholonomního robota. Tento podvozek ve dvourozměrném prostoru dovoluje pohyb pouze po jediné ose vpřed a vzad, jeho zadní kola nejsou otáčivá, a ty přední mají schopnost otáčet se pouze do omezeného úhlu. To mu dává možnost otáčet celým vozidlem pouze při pohybu tak, že svým pohybem



Obrázek 2.2: Příklad neholonomního objektu s Ackermanovým řízením (auto)

opisuje kružnici, jejíž poloměr má minimální hodnotu danou úhlem otáčení a vzdáleností kol.

Konfiguraci auta ve zjednodušené dvourozměrné projekci prostoru lze vyjádřit jako trojice $x = (x_1, x_2, x_3)$, kde x_1 a x_2 určují pozici v prostoru a x_3 úhel jeho natočení. Rychlost auta je relevantní pro konfiguraci, protože neovlivňuje trajektorii auta. Pro auto jedoucí vpřed rychlostí v a úhlovou rychlostí ω, pak můžeme psát diferenciální rovnici pro vektor konfigurace 2.2.

$$(x_1', x_2', x_3') = v(cos(x_3), sin(x_3), 0) + \omega(0, 0, 1) \tag{2.2}$$

Tato rovnice je ve skutečnosti shodná s diferenciálním řízením, pouze používá jiné proměnné. Ackermanovo řízení totiž vyžaduje přidání podmínky omezující rychlost otáčení v závislosti na rychlosti jízdy: $|\omega| < |v/r_{min}|$, kde r_{min} je poloměr kružnice rotace robota při maximálním otočení kol.

2.2.3 Auto s přívěsem

Často narážíme i na situaci, kdy auto (s Ackermanovým řízením) táhne přívěs (na obrázku 2.3). Tvoří se tak soustava dvou a více objektů, jejíž dvojice mají společný bod (osu) v místě napojení. Kolem této osy se mohou vzájemně otáčet, maximální úhel otočení je omezen hodnotou, kde narazí přívěs do táhnoucího auta. Zatímco auto má možnost pohybu (vpřed, vzad a otáčení předními koly), přívěs se řídí pouze pohybem auta (má jen pevnou nápravu bez motoru). Přívěsů může být i více a nemusejí dokonce být v řadě za sebou. [3]

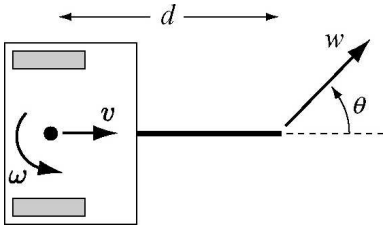
Problém řízení s přívěsem je složitější než řízení auta samotného. Předpokládejme, že přívěs je zapojen za autem. Při jízdě rovně vpřed se zdá být vše stejné, jako by žádný přívěs nebyl (přívěs je ustálen v rovné poloze). Při zatočení koly auta se náprava přívěsu bude blížit ke středu otáčení (ustálí se v poloze takové, že osa kol nápravy také směřuje do středu otáčení, je k němu tedy blíže).

Při couvání auta je situace podobná. Pouze s tím rozdílem, že se nejedná o konvergující stav, naopak od něj diverguje. Při plánování lze tento rozdíl jednoduše vyřešit (auto potřebuje couváním zatočit vpravo, proto nejdřív couvne kousek vlevo, tak aby se dostal s přívěsem do potřebného úhlu, následně malými změnami koriguje divergenci). Avšak při uskutečnění tohoto pohybu robotem je třeba zvýšená opatrnost.

Formálně můžeme konfiguraci auta s přívěsem popsat jako vektor $x = (x_1, x_2, x_3, x_4)$, kde x_1, x_2, x_3, mají stejný význam pro Ackermanovo řízení vozidla a x_4 je úhel otočení

Obrázek 2.3: Auto s přívěsem

přívěsu. Pro každý další přívěs je potřeba do konfigurace přidat jednu hodnotu určující úhel přívěsu. Vzdálenost středu přívěsu od bodu, kde je připojen, můžeme pro zjednodušení prohlásit za konstantní. Pak nás už jen zajímá úhlová rychlost přívěsu při jízdě, z ní už jednoduše vypočítáme polohu přívěsu.



Obrázek 2.4: Model přívěsu

Přívěs na obrázku 2.4 je tažen rychlostí w pod úhlem Θ se vzdáleností od středu k úchytu přívěsu d. Nás zajímá, jakou rychlostí v pojede tažený přívěs a s jakou úhlovou rychlostí ω (2.3) bude otáčen kolem své osy. Následně z toho doplníme diferenciální rovnici pro x_4 (2.4).

$$v = w\cos\Theta, \omega = \frac{w}{d}\sin\Theta \tag{2.3}$$

$$x_4' = \omega = \frac{w}{d}\sin\Theta \tag{2.4}$$

2.2.4 Další neholonomní objekty

Existuje i mnoho dalších druhů objektů, které jsou zařazeny mezi neholonomní. Můžeme sem řadit například chodící roboty. Ti používají k pohybu nohy inspirované přírodou. Můžeme je rozdělit podle počtu nohou, různý počet nohou dělá velké rozdíly. Na rozdíl od kolových robotů mají chodící roboti největší problém s rovnováhou a nedokáží vyvinout velké rychlosti na rovinách, zato však dokáží překonat i mnohem horší terény, na kterých by jiné kolové systémy zklamaly.

Nemůžeme opomenout ani létající roboty, které jsou v dnešní době stále více viditelné. Inspiraci přebírají z již existujících pilotovaných strojů. Díky vysoké stabilitě a bezpečnosti

Obrázek 2.5: Quadrocopter

je amatéry rozhodně nejoblíbenějším strojem quadrocopter (na obrázku 2.5). Jeho lehká konstrukce a čtyři motory s vrtulemi zajišťují snadné řízení a stabilitu v porovnání s klasickým vrtulníkem. Quadrocopter se může pohybovat všemi směry, ale nemůže se libovolně otáčet kolem svých os, při větším náklonu začne padat.

Největší tlak v tomto oboru byl vyvíjen právě pro vojenská použití, vznikly tak mnohá nová a nečekaná řešení. Robotická bezpilotní letadla dnes zkoumají nepřátelské území a získávají množství důležitých informací (odstranění kokpitu zjednodušilo a výrazně odlehčilo konstrukci, navíc neriskuje ztrátu na životech, pokud je průzkumné letadlo sestřeleno). Také se zde setkáme s inteligentními naváděnými raketami a odpovědí automatickými protiraketovými systémy.

Mezi neholonomní objekty patří i vesmírné lodě, sondy, raketoplány a rakety. Plánování pro tyto stroje má mnohem větší následky. Přesný a krátký zážeh na správném místě dokáže překonat vzdálenosti, které si člověk jen stěží dokáže představit.

Objekty ve vesmíru se nepohybují jen tak jak chtějí, vesmírnou loď není možné poslat přímou cestou na měsíc jako letadlo do jiného státu. Objekty se pohybují po eliptických oběžných drahách, pokud chceme dostat přistávací modul na Měsíc, je třeba plánovat přesnou trajektorii a tím ušetřit cenné palivo na návrat. Pro cesty mimo oběžné dráhy kolem Země se například používá technika gravitačního katapultu aplikovatelná na Měsíc. Využívá se efektu, který nastává při průletu nad povrchem tělesa. V pohledu relativním k tělesu náš objekt pouze změní směr, jelikož ale pouze prolétá, jeho oběžná dráha okolo tělesa, které ve skutečnosti obíhá, je změněna i rozměrově.

2.2.5 Vybrané neholonomní objekty pro applet

Výběr neholonomních objektů pro applet ovlivňuje použití pravděpodobnostních algoritmů. Tyto algoritmy jsou vhodné pro objekty, jenž se pohybují v omezeném prostředí s předem známými překážkami. Proto byly pro implementaci vybrány neholonomní objekty s diferenciálním podvozkem, objekty s Ackermanovým řízením a navíc vozidla s přívěsem.

2.3 Prostředí

Prostředí, neboli okolí ve kterém se robot pohybuje, je příliš obsáhlé a senzory skutečného robota jej nedokáží zachytit bez chyb. I kdyby se teoreticky podařilo dosáhnout přesného vnímání prostředí takového, jaké je, měli bychom nekonečné množství dat. Čím větší dostaneme model, tím je pomalejší výpočet problémů. Složitosti výpočtů bývají velmi často větší než lineární. Proto je důležité najít pokud možno co nejjednodušší, ale dobře použitelnou abstrakci prostředí.

Abychom mohli vůbec provádět akt plánování cesty, musíme vytvořit jistý abstrakt prostředí. Model je pak uložen v paměti robota. Robot tento abstrakt vytváří a upravuje podle toho, co vidí (snímá svými senzory), nebo co zjistil od ostatních robotů v systému či jiného zařízení (satelit, mapa stažená z internetu atd.). Na tomto abstraktu pak provádí samotné plánovaní, podle něj pak robot uskuteční svůj pohyb.

Pro tvorbu mapy prostředí se využívá různých senzorů (sonar, kamera, ...). Ze získaných dat a aktuální polohy robota je postupně skládána mapa okolí. Poloha robota je tedy velmi důležitým faktorem, pro získávání polohy se používá metoda triangulace nebo trilaterace, pomocí GPS nebo vysílačů umístěných v okolí.

Prostředí můžeme dělit podle několika faktorů: podle časové množiny (spojité a diskrétní), podle proměnlivosti v čase (statické a dynamické), podle předvídatelnosti (deterministické a nedeterministické) a další možná dělení. [15]

2.4 Pracovní a konfigurační prostor robota

Pracovní prostor je euklidovský n-rozměrný prostor, ve kterém se robot pohybuje a plní svůj úkol. Pracovní prostor definuje objekty a překážky v prostředí nebo omezující podmínky pro pohyb robota. Ty můžeme rozdělit na statické a dynamické. U statických objektů robot předpokládá že se v čase nemění, tedy při hledání cesty bude jistě vědět kudy tuto překážku může obejít. Dynamické objekty nemusejí být předvídatelné, proto je robot musí analyzovat až v jejich blízkosti. Robot v prostoru má svou polohu a směr otočení, případně polohy ramen, otočení kol a pod. Tyto hodnoty v jediném vektoru definují konfiguraci robota. Konfigurace robota tedy jednoznačně určuje polohu a orientaci robota.

Všechny konfigurace robota definují konfigurační prostor C. V konfiguračním prostoru C lze vymezit oblast C_{free} volný konfigurační prostor, kde jsou konfigurace robota přípustné, tedy nekoliduje z žádnou překážkou v pracovním prostoru a zároveň splňuje všechny omezující podmínky. Protiklad C_{obs} naopak robot nesplňuje alespoň jednu podmínku nebo koliduje s některou z překážek. Pak platí $C = C_{free} \cup C_{obs}$ a $\emptyset = C_{free} \cap C_{obs}$.

Pro zjednodušení je možné vytvořit volný konfigurační prostor expanzí překážek z pracovního prostoru, a to o poloměr kružnice opsané použitého robota. Pak můžeme konfigurace robota uvažovat jako množinu bodů, jenž nenáleží žádné z těchto expandovaných překážek. Dojde tím ke zjednodušení výpočtů kolizí a tím i ke zjednodušení hledání cesty robota.

Pro zdárné nalezení cesty je navíc nutné, aby počáteční konfigurace a cílová konfigurace ležela ve volném konfiguračním prostoru nakonec i s celou cestou.

2.5 Kolize s překážkami

Robot nalezl překážku ve scéně a uložil jí do mapy, upravil tak svůj pracovní prostor. Otázkou nyní je, jak zjistit, kam může až dojet, než narazí do této překážky, aby na základě této znalosti mohl plánovat cestu za svým cílem.

Mějme překážku definovanou jejími hraničními stěnami, pro výpočet kolize nám pak stačí objekt znázorňující tvar robota (fyzikální model). Zjistíme, zda-li mají objekty vzájemně průnik. Existuje-li nějaký průnik, nebo jeden objekt z objektů obaluje druhý, konfigurace robota spadá do kolizního konfiguračního prostoru a naopak. Tato metoda však není efektivní a ve složitějších a rozsáhlejších prostorech je příliš náročná, navíc je třeba počítat kolize pro každou konfiguraci robota. Tento problém alespoň z části řeší rozdělení

scény a použití obalových těles. Více informací o těchto metodách naleznete v knize „Real-Time Collision Detection“ [4].

2.5.1 Rozdělení scény

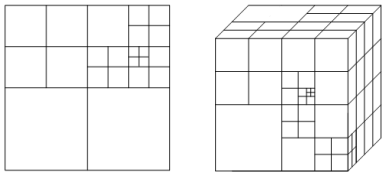
Základní myšlenkou těchto metod je rozdělení scény do buněk, každý objekt ve scéně pak spadá do jedné buňky, v jedné buňce může být více objektů. Kolize mezi objekty jsou pak počítány pouze s objekty ve stejné nebo sousední buňce. Metody rozdělení scény jsou podrobněji popsány v knize [4].

Rozdělení do mřížky

Rozdělení do mřížky je nejjednodušší metoda z hlediska implementace. Rozděluje prostor do stejně velkých buněk stejného tvaru. K určení náležitosti objektu do buňky stačí zaokrouhlení pozice. Není však příliš robustní. V případě výskytu kombinace malých a velkých objektů řádově větších než malé objekty, není možné zajistit ideální velikost buněk. Buďto budou příliš malé a velké objekty zaberou moc buněk. Nebo budou příliš velké a malých objektů bude mnoho v jediné buňce. Dalším problémem je rozsáhlost prostoru, toto rozdělení je pak náročné na paměť. Tyto problémy řeší například hierarchické rozdělení do mřížky, kde je použito hned několik mřížek s různými velikostmi buněk. Objekty jsou pak přiřazovány do buňky nejen na základě pozice, ale i jejich velikosti.

Rozdělení do stromu

Myšlenka tohoto rozdělení je taková, že celý prostor reprezentuje jedna buňka – kořen stromu. Pokud počet objektů v buňce překročí předem určenou hranici, buňka je rozdělena do několika menších buněk (v n rozměrech na 2^n buněk s poloviční délkou hrany viz obr. 2.6). Stejné pravidlo platí i pro nové buňky. To zaručuje omezení výskytu velkého počtu objektů v jediné buňce a zároveň prázdné oblasti nejsou zbytečně vyplněny buňkami.



Obrázek 2.6: Hierarchické rozdělení scény (quadtree vlevo, octree vpravo).

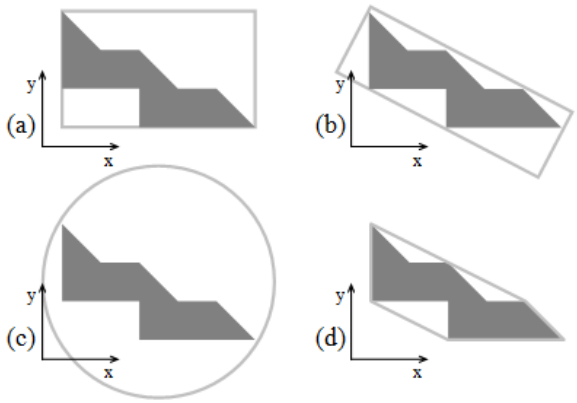
2.5.2 Obalová tělesa

Dalším možným ulehčením výpočtu kolizí jsou tzv. obalová tělesa. Myšlenka zní tak, že složitá a výpočetně náročná tělesa lze obalit jednoduchým tělesem, ke složitým výpočtům přistoupit pouze tehdy, kdy kolidují jednoduchá obalová tělesa. Tato metoda sice přidává práci o výpočet kolize obalových těles, ale ve velké míře se stává že obalová tělesa nekolidují a ušetří se tím složitější výpočty. Až na některé výjimky tato metoda výrazně urychluje výpočet. Metody obalových těles jsou podrobněji popsány v knize [4].

Nutno je však zvolit vhodný typ obalových těles. Měly by mít co nejjednodušší výpočet průsečíků, co nejtěsněji přiléhat k objektu, mít možnost snadné rotace, posuvů a jiných

operací, ale také nezabírat moc paměti. V nejlepším případě tak, že poměr chybných odhadů obalových těles je co nejmenší a zároveň výpočet kolize mezi obaly mnohem jednodušší než mezi samotnými objekty.

Nejčastějšími používanými reprezentanty jsou koule, kvádry (podle orientace dále dělíme na orientované a pevné) a konvexní obálka (viz obrázek 2.7).



Obrázek 2.7: Příklady obalových těles: (a) AABB, (b) OBB, (c) koule, (d) konvexní obálka

Osově zarovnané kvádry (AABB)

Objekty jsou obaleny kvádrem (ve 2D obdélníkem), jehož hrany jsou rovnoběžné s osami prostoru. Každý kvádr pro definici potřebuje střed, délku, šířku a výšku. Důsledkem toho výpočet kolize mezi obalovými tělesy spočívá pouze v porovnání a jednoduchých součtů. Tedy je velmi rychlý pro výpočty, avšak mnohdy nijak zvlášť nepřiléhá k objektům (dlouhé a úzké) a dochází častěji k falešným výpočtům složitější kolize.

Orientované kvádry (OBB)

Použití kvádrů tentokrát libovolně orientovaných přináší užitek pro dlouhé úzké objekty. Pro jednoznačnou definici kvádru je třeba navíc i jeho orientace v prostoru. Výpočet je zde složitější a vyžaduje práci s lineárními rovnicemi. Zjištění průsečíků je však pořád jednodušší než kolize dvou komplikovaných objektů. Důležité je dobré natočení kvádru tak, aby k objektu přiléhal co nejlépe. U objektů složitějších tvarů bude problém nepřiléhání přetrvávat.

Konvexní obálka

Další častou možností je použití tzv. konvexní obálky. Pro její jednoznačnou definici postačuje množina vektorů určujících směr a posun rovin určujících stěny obálky. Celý objekt je pak uschován za touto obálkou. K pravému výpočtu kolize dochází, až když se nějaká část z cizího tělesa dotýká obalu. Konvexní obálka velmi dobře přiléhá k tělesům přibližně konvexního tvaru. Například stupňovitá pyramida s obalovým jehlanem.

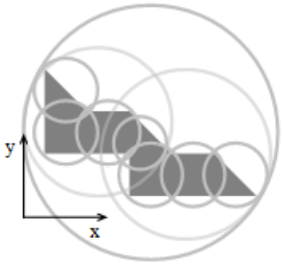
Koule

V neposlední řadě sem patří obalová koule (ve 2D kruh). Definuje ji její střed a poloměr. Pro výpočet kolize mezi obalovými tělesy pak postačuje vzdálenost mezi koulemi a součet jejich poloměrů v nerovnici. Při výpočtu se nepoužívá odmocnina, zato druhá strana Pythagorovy rovnice se umocní. Na moderních architekturách není výpočet vzdálenosti tak znatelný. Výpočet je rychlý, avšak koule mnohdy nepřiléhají k tělesům.

2.5.3 Hierarchie obalových těles

Mějme však objekt, jenž je větší a mnohem složitější, a menší objekty pohybující se uvnitř nebo v jeho blízkosti. V takovém případě výše zmíněná metoda nepomůže, naopak zvyšuje výpočetní nároky. Naskýtá se však docela jednoduché řešení použití hierarchicky uspořádaných obalových těles. Složitější a větší objekty se rozdělí na menší části, ty jsou obaleny jednoduchými tělesy (koule, AABB nebo OBB). Skupiny blízkých těles pak obalují větší tělesa (jednoduchá pro výpočet), tím vytvářejí stromy obalových těles (viz obrázek 2.8). Každý uzel ve stromě obaluje všechny své potoky. Listy stromu jsou části objektů a každý uzel představuje obalové těleso. Celý prostor je zabalen do lesu hierarchicky uspořádaných obalových těles. [4]

Pro zjištění kolizí mezi objekty je třeba začít od kolizí kořenových obalových těles obou objektů. Kolidují-li, rekurzivně přecházíme do nižších vrstev až ke kolidujícím listům stromu, pak teprve počítá kolize mezi danými částmi objektů.



Obrázek 2.8: Hierarchie obalových těles.

Kapitola 3

Hledání cesty

Pro hledání cest existuje mnoho různých algoritmů. V této kapitole jich bude několik rozebráno. Zaměříme se především na pravděpodobnostní algoritmy. Zajímavý přehled metod pro hledání cesty je popsán v knize [9]. Dále byly informace čerpány z prací [7], [6], [13] a knihy [3].

Robot pro plánování cesty nutně potřebuje mapu prostředí, svůj aktuální stav, obsahující jeho pozici v mapě, a také cílový stav nebo nutnou podmínku pro splnění cíle. Na základě těchto znalostí by pak měl být schopen vytvořit plán, s něhož pomocí cíle dosáhne.

Při plánování cesty každou možnou situaci robotu nazýváme stav a označujeme ji symbolem x. Množinu všech takových stavů nazýváme stavový prostor a označujeme jej symbolem X. Aplikací možné akce robota u na jeho aktuální stav x se změní jeho stav na x' takový, že $x' = f(x, u)$, kde f je přechodová funkce robota. Množina všech akcí aplikovatelných na stav x funkcí f tvoří tzv. akční prostor označovaný jako $U(x)$. Formálně je tedy úkolem hledání cesty najít konečnou posloupnost akcí, kterými postupně transformujeme počáteční stav robotu do cílového stavu.

Nejkratší cesta grafu mezi počátečním uzlem a koncovým uzlem je cesta mezi těmito uzly taková, že součet vah všech hran cesty je nejmenší možný. Takových nejmenších cest může být v grafu více.

Exaktní (úplné) plánování cesty

Metoda plánování cesty je exaktní (také označovaná jako kompletní nebo úplná) právě tehdy, když garantuje nalezení cesty mezi počáteční a cílovou konfigurací za předpokladu, že tato cesta existuje.

Pravděpodobnostně úplné plánování cesty

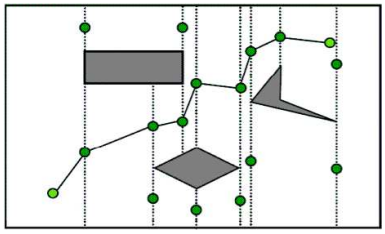
Metoda plánování cesty je pravděpodobnostně úplné, právě když existuje-li cesta, algoritmus má určitou nenulovou pravděpodobnost nalezení řešení. Do této třídy spadají například pravděpodobnostní algoritmy.

3.1 Metody diskretizace prostoru

Tyto metody z pracovního prostoru vybírají konečnou množinu volných konfigurací, z nich tvoří graf s možnými přechody. Na něm pak aplikací například Dijkstrova algoritmu je možné najít cestu, která dovede robota do cílové konfigurace. Několik základních metod je zde blíže popsáno, podrobnější informace se nacházejí v [11].

3.1.1 Exaktní rozklad do buněk

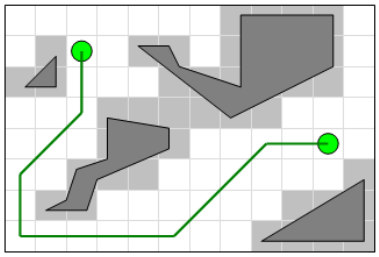
Volný konfigurační prostor je rozdělen do množiny nepřekrývajících buněk, často do buněk lichoběžníkového (viz obrázek 3.1) nebo trojúhelníkového tvaru tak, aby zjednodušili výpočet míst kudy může robot přecházet mezi dvěma buňkami. Například pro jednoduchost přes střed společné strany buněk. Tyto místa přechodů spolu se startovní a cílovou konfigurací pak slouží jako uzly grafu. Každý takový uzel je propojen hranou s ostatními uzly náležícími buňkám, na kterých leží. Každá pak buňka tvoří v grafu kliku. Tento graf je pak použit pro nalezení cesty prostorem. [9]



Obrázek 3.1: Příklad algoritmu rozkladu na buňky (lichoběžníkové buňky).

3.1.2 Aproximativní rozklad do buněk

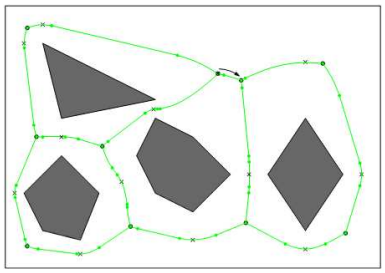
Rozkládá volný konfigurační prostor do buněk stejného tvaru (pro dvourozměrný prostor jsou to trojúhelníky, čtverce nebo šestiúhelníky, ve třech rozměrech pak krychle). Každou buňku, která koliduje s některou z překážek nebo nesplňuje nějakou podmínku, označíme jako obsazenou (viz obrázek 3.2. Všechny neobsazené buňky pak tvoří uzly grafu a jsou spojeny hranami se sousedními buňkami. Úspěšnost této metody závisí na velikosti použitých buněk. To pak znamená, že tato metoda není úplná. Při použití příliš malých buněk se zvyšuje náročnost metody. [9]



Obrázek 3.2: Příklad aproximativního rozkladu do buněk, čtvercové buňky

3.1.3 Voronoiův diagram

Trajektorie robota opisuje hrany tzv. Voronoiova diagramu (viz obrázek 3.3). Tyto hrany jsou definovány tak, aby v každém bodě maximalizovali vzdálenost od nejbližší překážky. Pak každý bod Voronoiova diagramu má stejnou vzdálenost minimálně ke dvěma nejbližším překážkám (vyvážený stav, po oddálení od jedné se musí přiblížit ke druhé). Každý uzel grafu je bod, jehož alespoň tři nejbližší překážky mají k němu stejnou nejmenší vzdálenost. Po převodu Voronoiova diagramu na graf je pak Nalezení cesty možné například Dijkstrovým algoritmem. [9]



Obrázek 3.3: Voronoiův diagram, příklad

3.2 Pravděpodobnostní algoritmy

Pravděpodobnostní algoritmy mají velmi široké pole využití a lze je použít pro všechny roboty pracující ve známém prostoru. Využití nacházejí právě ve vícerozměrných konfiguračních prostorech, jako například u robotických ramen. [8]

Základní myšlenkou pravděpodobnostních algoritmů je vzorkování volného konfiguračního prostoru a následného vytvoření grafu, nad kterým probíhá prohledávání (například pomocí Dijkstrova algoritmu). Pravděpodobnostní algoritmy jsou pravděpodobnostně úplné, to znamená, že pokud existuje cesta, algoritmus má určitou nenulovou pravděpodobnost nalezení. Doba výpočtu tedy zvyšuje pravděpodobnost, že nějaké řešení algoritmem najdeme.

Pravděpodobnostní metody dělíme podle toho, zda-li vytvořený graf zachycující daný prostor můžeme využít opakovaně pro různé počáteční a koncové stavy či nikoliv, na tyto skupiny:

- Jednodotazové – Je vytvořen graf pouze pro danou kombinaci počáteční a cílové konfigurace. Při dalším hledání je nutné vytvořit graf nový. Sem patří metody RRT a EST.
- Vícedotazové – Je vytvořen graf vhodný pro libovolnou dvojici konfigurací robota. Při dalším hledání jej lze znovu použít, není potřeba znovu konstruovat graf. Sem patří například metoda PRM.
- Kombinované – Sem patří například algoritmus SRT, který je možné použít jako vícedotazový, sám ovšem interně využívá jednodotazové algoritmy.

3.2.1 PRM – Probabilistic roadmaps

PRM je základním a nejjednodušším pravděpodobnostním algoritmem. PRM je vhodný pro vícerozměrné problémy a neholonomní objekty ve statickém a předem známém prostředí. Tento algoritmus je vícedotazový a pracuje ve dvou fázích. [9]

V první fázi je vytvořen graf. Jako uzly jsou vybírány náhodné konfigurace z volného konfiguračního prostoru a následně pak spojovány hranami tak, aby hrany značily možný přechod robotů mezi konfiguracemi (přechod musí být bezkolizní a musí splněny všechny podmínky pohybu robota). Graf by pak měl pokrýt vzorky celý volný konfigurační prostor.

Ve druhé fázi do vytvořeného grafu algoritmus napojí startovní a cílovou konfiguraci a následně grafem hledá cestu. Graf v předchozí fázi byl vytvořen nezávisle na startovní a cílové konfiguraci, na kterých již druhá fáze závisí. Pro opakované hledání s jiným zadáním tedy stačí zopakovat jen druhou fázi (vícedotazová metoda). Hledání cesty bývá realizováno například Dijkstrovým algoritmem.

Vytvoření grafu

První fáze algoritmu vytváří graf $G = (V, E)$, kde V je množina uzlů reprezentujících náhodně vybrané volné konfigurace a $E \subseteq V \times V$ je množina hran grafu, tedy možných bezkolizních pohybů robota mezi konfiguracemi. Pro spojování uzlů je důležitá *spojovací funkce* Δ a *vzdálenostní funkce dist*.

Vzdálenostní funkce $dist : C \times C \to \mathcal{R}_0^+$ určuje (euklidovskou) vzdálenost mezi dvěma konfiguracemi. Používá se při hledání nejbližších sousedů.

Spojovací funkce $\Delta : C \times C \to M \cup \{NIL\}$ hledá možný pohyb robota konfiguračním prostorem mezi dvěma konfiguracemi, kde M je množina všech možných pohybů robota. Najde-li se takový pohyb $m \in M$, který však nezapříčiní žádnou kolizi, ani porušení omezujících podmínek, funkce vrací m, jinak vrací NIL.

Algoritmus 3.1 na počátku vytvoří prázdný graf. Začne přidáváním uzlů s náhodnými konfiguracemi. Přidávají se pouze uzly, které splňují podmínky a nekolidují s překážkami. Každý uzel nese informaci s jeho konfigurací. Přidávání uzlu pokračuje, dokud jejich počet nedosáhne zadané hodnoty n. Graf pak obsahuje n uzlů a žádnou hranu.

Nyní algoritmus přejde ke generování hran. Pro každý uzel q v grafu najde k nejbližších uzlů. Se všemi z nich (bez těch, se kterými již je spojen hranou) otestuje, zdali je možné provést pohyb nebo jednoduchý sled pohybů (funkce Δ). Podaří-li se, přidá do grafu hranu spojující tyto dva uzly. Tím algoritmus graf zaplní sítí hran. Na tomto grafu pak probíhá druhá fáze PRM.

Hodnoty n a k je potřeba nastavovat s rozvahou. Čím větší je n, tím větší je i úspěšnost nalezení cesty. Velké a složité prostory potřebují větší n, ale s ním roste čas generování stromu.

Nalezení cesty v grafu

Druhá fáze se již může opakovat na stejném grafu s různými vstupními konfiguracemi. Vstupem algoritmu je graf vytvořený v první fázi, startovní a cílová konfigurace robotu a počet k nejbližších uzlů pro jejich napojení do grafu.

Zaprvé algoritmus 3.2 napojí startovní a cílovou konfiguraci do grafu přidáním dvou uzlů a generováním hran podobným způsobem jako v první fázi. Hodnota k se oproti první fáze může lišit. Do grafu je však přidána pouze první hrana, splňující podmínku. Graf by

Algoritmus 3.1 Algoritmus pro vytvoření náhodného grafu PRM.

Vstup: n,k
n: počet uzlů výsledného grafu
k: počet nejbližších sousedů uzlu pro napojení
Výstup: G: graf $G = (V, E)$

```
1: V ← ∅
2: E ← ∅
3: while |V| < n do
4:    repeat
5:       q ← náhodná konfigurace z C
6:    until q je volná konfigurace
7:    V ← V ∪ {q}
8: end while
9: for all q ∈ V do
10:   N_q ← k nejbližších sousedů podle vzdálenostní funkce dist
11:   for all q' ∈ N_q do
12:      if (q, q') ∉ E and Δ(q, q') ≠ NIL then
13:         E ← E ∪ {(q, q')}
14:      end if
15:   end for
16: end for
```

měl být tedy rozšířen o dva uzly a dvě hrany. To však není zaručeno, pokud se stane, že start nebo cíl není napojen do grafu, znamená to, že vygenerovaný graf v první fázi nepokrývá dostatečně prostor, cestu pak nebude možné najít.

Na tomto grafu pak algoritmus provede výpočet cesty. Nejčastěji se používá Dijkstrův algoritmus nebo A*. Pokud algoritmus nalezne cestu, vrátí ji na výstup. Může se stát, že cesta v grafu neexistuje, to neznamená, že se robotovi nemůže podařit nikdy cestu najít při opakovaných výpočtech. Vždy je možné opakovat první fázi a vytvořit tak hustší graf s více uzly, nebo jiným způsobem zakomponovat do již existujícího grafu nové uzly. Tím se zvedne pravděpodobnost nalezení cesty.

3.2.2 RRT – Rapidly-exploring Random Trees

V minulé kapitole byl rozebrán vícedotazový algoritmus PRM, tato kapitola se věnuje tentokrát jednodotazovému algoritmu. Tím je RRT [10]. Tento algoritmus pracuje se dvěma stromy, jejich kořeny jsou startovní a cílová konfigurace. Graf tedy nelze aplikovat opakovaně pro různá zadání, musí být vytvořen nový (proto jednodotazový).

Vytvoření stromů

První fáze (viz algoritmus 3.3) začíná vytvořením stromů pro startovní a cílovou konfiguraci, právě tyto konfigurace použije jako kořeny stromů. Následně tyto stromy rozšiřuje.

Pro každý strom provede n pokusů o přidání nového uzlu do stromu. Při každém pokusu vygeneruje náhodnou konfiguraci q_{rand}. Najde k ní nejbližší uzel v grafu q_{near} a na úsečce mezi těmito konfiguracemi zvolí bod q_{new} tak, že je vzdálen od q_{near} na vzdálenost $step\_size$. Je-li úsečka kratší bere se v potaz maximální možná vzdálenost (viz algoritmus 3.4). Pokud takový bod q_{new} existuje a zároveň je možné se do něj dostat z q_{near} bez kolize a se splněním

Algoritmus 3.2 Algoritmus pro nalezení cesty v grafu PRM.

Vstup: q_{init},q_{goal},k,G
q_{init}: startovní konfigurace
q_{goal}: cílová konfigurace
k: počet nejbližších uzlů pro napojení
G: graf $G = (V, E)$ vytvořen pomocí algoritmu 3.1
Výstup: P: cesta z q_{init} do q_{goal} v G

$N_{q_{init}} \leftarrow k$ nejbližších sousedů q_{init} podle vzdálenostní funkce $dist$
$N_{q_{goal}} \leftarrow k$ nejbližších sousedů q_{goal} podle vzdálenostní funkce $dist$
$V \leftarrow V \cup \{q_{init}, q_{goal}\}$
$q' \leftarrow$ nejbližší sousední uzel uzlu q_{init} vybraný z $N_{q_{init}}$
repeat
  if $\Delta(q_{init}, q') \neq NIL$ **then**
    $E \leftarrow E \cup \{(q_{init}, q')\}$
  else
    $q' \leftarrow$ další nejbližší sousední uzel uzlu q_{init} vybraný z $N_{q_{init}}$
  end if
until uzel q_{init} byl úspěšně napojen nebo $N_{q_{init}} = \emptyset$
$q' \leftarrow$ nejbližší sousední uzel uzlu q_{goal} vybraný z $N_{q_{goal}}$
repeat
  if $\Delta(q_{goal}, q') \neq NIL$ **then**
    $E \leftarrow E \cup \{(q_{goal}, q')\}$
  else
    $q' \leftarrow$ další nejbližší sousední uzel uzlu q_{goal} vybraný z $N_{q_{goal}}$
  end if
until uzel q_{goal} byl úspěšně napojen nebo $N_{q_{goal}} = \emptyset$
$P \leftarrow$ nejkratší cesta z q_{init} do q_{goal} v G
if $P \neq \emptyset$ **then**
  return P
else
  return $failure$
end if

všech podmínek, pak tento bod přidá do stromu a spojí jej hranou (q_{near}, q_{new}). Po ukončení n pokusů spojení pak dostaneme vygenerovaný strom. Celkem tedy dva oddělené stromy, jež budou v druhé fázi spojovány.

Důležité je zvolení správných parametrů n a $step\_size$. Při zvolení nízkých hodnot n se stromy nemusejí setkat, naopak velkou hodnotou zvýšíme čas výpočtu na neúnosnou míru. Také délka kroku $step\_size$ ovlivňuje výpočet. Čím větší má hodnotu, tím větší prostor zabere strom, jelikož určuje délku hran, v úzkých a klikatých prostorech je nutné volit tak, aby robot dokázal projít. Tuto hodnotu je možné i dynamicky měnit během vytváření stromů.

Spojení stromů

Druhá fáze algoritmy RRT bude spojovat oba vygenerované stromy do jediného grafu. Výstupem jsou oba stromy a počet pokusů o spojení mezi stromy.

Algoritmus 3.5 provede l pokusů o spojení. Při každém pokusu vygeneruje náhodnou

Algoritmus 3.3 Algoritmus pro konstrukci stromů RRT.

Vstup: q_0, n
q_0: kořen stromu (počáteční uzel v případě T_{init}, koncový uzel v případě T_{goal}
n: maximální počet pokusů o rozšíření stromu
Výstup: $T = (V, E)$: strom s kořenem q_0
1: $V \leftarrow \{q_0\}$
2: $E \leftarrow \emptyset$
3: **for** $i = 1$ to n **do**
4: $q_{rand} \leftarrow$ náhodně vybraná volná konfigurace
5: $extendRRT(T, q_{rand})$
6: **end for**
7: **return** T

Algoritmus 3.4 *extendRRT* algoritmus pro spojení uzlů RRT.

Vstup: T, q
T: strom z algoritmu RRT $T = (V, E)$
q: konfigurace, ke které bude strom expandovat
Výstup: q_{new}: nový uzel s konfigurací nebo NIL
1: $q_{near} \leftarrow$ nejbližší uzel z V ke konfiguraci q
2: $q_{new} \leftarrow$ bod na úsečce (q, q_{near}) ve vzdálenosti $step\_size$ od q_{near}
3: **if** $\Delta(q_{near}, q_{new}) \neq NIL$ **then**
4: $V \leftarrow V \cup \{q_{new}\}$
5: $E \leftarrow E \cup \{(q_{near}, q_{new})\}$
6: **return** q_{new}
7: **end if**
8: **return** NIL

konfiguraci q_{rand} a vytvoří uzel $q_{new,1}$ k prvnímu stromu obdobně jako při vytváření grafu, posune jej na vzdálenost $step\_size$ k nejbližšímu uzlu q_{near} (pokud již není dost blízko). Následně se pokusí spojit nový vygenerovaný uzel s druhým stromem tak, že vytvoří $q_{new,2}$ na úsečce mezi $q_{new,1}$ a $q_{near,2}$ (nejbližší uzel druhého stromu), a obdobně provede posuv. Podaří-li se, algoritmus končí jediným stromem. Pak existuje jediná cesta grafem mezi uzly q_{init} a q_{goal}. Nepodaří-li se spojení stromů, algoritmus zkusí spojení začít druhým stromem znovu (operace *swap* zamění stromy). Pokus je opakován l krát, nepodaří-li se ani poté spojení stromů, algoritmus končí neúspěchem.

3.2.3 EST – Expansive-Spaces Trees

EST je také jednodotazový algoritmus. Pracuje podobně jako RRT. Používá se jeho dvoustromová nebo jednostromová verze. V případě jednostromové je generován pouze jeden strom z cíle nebo ze startovní konfigurace. [9]

Vytvoření stromu

Algoritmus 3.6 provádí generování stromů, začíná se stromy (nebo stromem) s kořeny ve startovní a cílové konfiguraci. Pro každý strom pak provede n pokusů o přidání uzlu. Oproti RRT již nepoužije k vytvoření uzlu rovnoměrné rozložení, místo toho použije pravděpodobnostní funkci π_T pro vybrání již existující volné konfigurace ve stromě. K této náhodné

Algoritmus 3.5 Algoritmus spojující stromy RRT.

Vstup: T_1, T_2, l
T_1: první RRT strom $T_1 = (V_1, E_1)$
T_2: druhý RRT strom $T_2 = (V_2, E_2)$
l: maximální počet pokusů o propojení stromů T_1 a T_2
Výstup: *merged* pokud je spojení úspěšné, jinak $failure$

```
for i = 1 to l do
    q_rand ← náhodně vybraná volná konfigurace
    q_new,1 ← extendRRT(T_1, q_rand)
    if q_new,1 ≠ NIL then
        q_new,2 ← extendRRT(T_2, q_new,1)
        if q_new,1 = q_new2 then
            return merged
        end if
        Swap(T_1, T_2)
    end if
end for
return failure
```

konfiguraci Q_{rand} algoritmus vytvoří v jeho okolí nový uzel q_{new} a přidá jej do grafu, pokud nekoliduje a je možné provést nekolizní pohyb robota mezi těmito konfiguracemi. Pokus o přidání uzlu opakuje, dokud počet pokusů nedosáhne n.

Výsledkem je pak strom nebo dva stromy (v případě dvoustromové verze).

Algoritmus 3.6 Algoritmus pro konstrukci stromů EST.

Vstup: q_0, n
q_0: kořen stromu
n: maximální počet pokusů o rozšíření stromu
Výstup: T: strom $T = (V, E)$ s kořenem q_0

```
V ← {q_0}
E ← ∅
for i = 1 to n do
    q_rand ← volná konfigurace vybraná s pravděpodobností π_T(q_rand)
    q_new ← náhodně vybraná konfigurace v okolí konfigurace q_rand
    if Δ(q_rand, q_new) ≠ NIL then
        V ← V ∪ {q_new}
        E ← E ∪ {(q_rand, q_new)}
    end if
end for
return T
```

Pravděpodobnostní funkce výběru uzlu π_T musí být zvolena tak, aby nedocházelo k převzorkování, největší nebezpečí hrozí u startu a cíle. To by výrazně snižovalo efektivitu algoritmu. V praxi se osvědčilo použití hustoty nashromážděných uzlů. Pravděpodobnost $\pi_T(q)$ je tím větší, čím méně je v okolí q jiných uzlů. V úvahu připadá i pořadí vytvoření uzlu, počet hran a vzdálenost k cíli.

Spojení

V případě jednostromové verze stačí zkontrolovat, zda je možné spojit cíl s některým z uzlů stromu. V případě dvoustromové verze je třeba obdobně jako u RRT provést spojení stromů. Při spojování využívá stejnou pravděpodobnostní funkci π_T. Algoritmus vybírá náhodně uzly a pokouší se je spojit s druhým stromem. Najde-li spojující hranu, existuje i cesta mezi startem a cílem.

3.2.4 SBL – Single-query, Bidirectional, Lazy collison checking

Hlavní myšlenka tohoto algoritmu spočívá v odložení kontroly kolizí přechodů až do doby poté, kdy je nalezena cesta. Přechody mezi konfiguracemi jsou do té doby považovány za proveditelné.

Například pomocí PRM je vygenerován graf bez použití kontroly splnitelnosti přechodů Δ, všechny přechody uspějí v přidání do stromu. Po nalezení cesty jsou přechody zkontrolovány, a jsou-li neproveditelné, přechod je odstraněn a hledá se nová cesta grafem, dokud není nalezena taková cesta, pro které jsou všechny přechody proveditelné.

Tento přístup má za následek rapidní pokles nutných kontrol přechodů, kontrolují se jen ty, které mohou vést k řešení cesty, na druhou stranu je algoritmus hledání cesty grafem zatížen větším počtem hran.

3.2.5 SRT – Sampling-Based Roadmap of Trees

Tento algoritmus stojí někde mezi jednodotazovým a vícedotazovým přístupem, lze jej použít oběma způsoby. Jeho přístup je založen na grafu (roadmap), který je tvořena pomocí kombinace metod algoritmů PRM a EST nebo RRT (informace čerpány z knihy [3]).

Vytvoření grafu

Jako u všech pravděpodobnostních algoritmů, je i u SRT nutné nejprve vytvořit graf (road-mapu). Základní graf je vytvořen pomocí PRM (několik uzlů s rovnoměrným rozložením pravděpodobnosti), jeho uzly však neskončí jako jednotlivé konfigurace, ale jsou z nich vygenerovány stromy. Startovní a cílový uzel jsou napojeny do grafu buď hned na začátku nebo až později s napojením do výsledného grafu. Každý uzel grafu je použit jako kořen a pomocí algoritmů RRT expand nebo EST expand je expandován. provádí se určitý počet expanzí.

Stromy ve výsledném lese jsou po vygenerování spojovány do grafu road-mapy. Algoritmus vyhledává možné přechody mezi stromy (viz algoritmus 3.7). Nejdříve zkouší každý strom spojovat s k nejbližšími stromy, pak s r náhodně vybranými stromy. Vzdálenost stromu je vzdálenost k těžišti stromu, tj. průměrná poloha. Spojování je provedeno výběrem několika bodů z jednoho stromu a následným pokusem o spojení s jejich nejbližšími sousedy z druhého stromu.

Je-li výsledný graf souvislý, pak cesta bude určitě nalezena. Hledání cesty mezi startovní a cílovou konfigurací je provedeno prohledávaním algoritmem (Dijkstrův algoritmus).

Vlastnosti SRT

Díky tomuto přístupu získává SRT několik vlastností, které stojí za zmínku. SRT se v případě nastavení počtu expanzí na 0 stává algoritmem PRM, naopak při nastavení počtu

vygenerovaných náhodných kořenů na 0 zůstanou jen startovní a cílový strom, z čehož vznikne původní algoritmus RRT nebo EST.

Protože je generováno hned několik nezávislých stromů, které jsou spojovány po dvojicích, tento algoritmus je možné jednoduše paralelizovat. Každý procesor může generovat jeden strom a následně pokoušet dva stromy spojit.

SRT má obecně dobré výsledky, díky množině vygenerovaných kořenů je prohledávání rozloženo rovnoměrně po celém prostoru.

Algoritmus 3.7 Algoritmus pro spojení stromů SRT.

Vstup: V_r, k, r
V_T: Seznam stromů pro napojení
k: počet nejbližších sousedních stromů pro napojení
r: počet náhodných stromů pro napojení
Výstup: G_T: graf $G_T = (V_T, E_T)$ složený z napojených stromů z V_r

$E_T \leftarrow \emptyset$
for all $T_i \in V_T$ **do**
  $N_{T_i} \leftarrow k$ sousedních a r náhodných stromů z V_T stromu T_i
  for all $T_j \in N_{T_i}$ **do**
    if T_i a T_j nejsou součástí jedné komponenty grafu G_T **then**
      $merged \leftarrow FALSE$
      $S_i \leftarrow$ seznam náhodně vybraných bodů z T_i
      for all $q_j \in S_i$ **and** $merged = FALSE$ **do**
        $q_j \leftarrow$ bod z T_j nejbližší z bodů q_i
        if $\Delta(q_i, q_j)$ **then**
          $E_T \leftarrow E_T \cup \{(T_i, T_j)\}$
          $merged \leftarrow TRUE$
        end if
      end for
      if $merged = FALSE$ **and** $MergedTrees(T_i, T_j)$ **then**
        $E_T \leftarrow E_T \cup \{(T_i, T_j)\}$
      end if
    end if
  end for
end for

3.3 Plánování cest pro neholonomní roboty

Existuje mnoho možností hledání cest pro neholonomní algoritmy. Možné jsou jak úpravy existujících algoritmů nebo dodatečná úprava nalezené cesty. Taková úprava probíhá ve třech krocích. [14]

První krok provede výpočet co nejkratší cesty pro holonomního robota s vhodně zvolenými parametry. Použít lze například pravděpodobnostní algoritmy.

Druhý krok provede transformaci nalezené cesty pro neholonomního robota tak, aby splňoval všechny nutné podmínky. K tomuto účelu je možno použít Reeds– Shepp křivky nebo Grid Search plánovače.

Transformace nahrazuje cestu z prvního kroku. Transformace probíhá iterativně nejprve pro celou cestu, nepodaří-li se úsek spojit jednoduchou nekolizní křivkou respektující neholonomního robota, úsek je rozdělen na dva poloviční a ty se transformují znovu. Tím se rekurzivně zanořují, dokud není nalezena nekolizní cesta.

Třetí krok následně optimalizuje transformovanou cestu. Optimalizace pracuje s cestou a zkoumá možnosti zkrácení cesty nebo jakékoli zkratky. Nalezenými zkratkami následně nahrazuje příslušné úseky cesty.

3.3.1 Dubinsovy křivky

Znázorňuje křivky trajektorie pohybu neholonomního vozidla typu auto. Často se používají i pro plánování tras letadel. Dubinsovy křivky totiž předpokládají, že se objekt pohybuje pouze jedním směrem po křivce (nemůže se v průběhu rozhodnout, že začne couvat) [9]. Tato trajektorie se skládá z posloupnosti úseků, kde každý úsek je označen písmeny: S – auto jede přímo (po přímce), L – auto otáčí naplno vlevo (opisuje kružnici s minimálním poloměrem, které dovolí Ackermanovo řízení) a R – auto zatáčí naplno vpravo (stejný poloměr jako v předchozím případě). Úseky na sebe plynule navazují, při přechodu není žádný zlom, auto se také otáčí plynule. [9]

Dubins ukázal že optimální cestou je vždy alespoň jedna z kombinací z množiny:

$\{LRL, RLR, LSL, LSR, RSL, RSR\}$.

Můžeme tedy předpokládat, že pro neholonomní roboty typu auto, pohybující se pouze vpřed, stačí testovat těchto 6 posloupností a vybrat tu nejkratší, která však nesmí kolidovat s překážkami.

3.3.2 Reeds – Shepp křivky

Pracují podobně jako Dubinsovy křivky. Pouze s tím rozdílem, že auto již může i couvat. Pro zjednodušení byly zavedeny symboly $|$ – pro otočení směru pohyby, S – jízda vpřed, C – auto zatáčí naplno do jednoho směru, při každém použitím C se směr změní na opačný (CC je LR nebo RL). Navíc přidání dolního indexu specifikuje přesný úhel otočení při provádění C ($C_{\frac{\pi}{4}}$ je otočení o 45°). Příklad křivky je na obrázku 3.4. [9]

Reeds a Shepp ukázali že optimální cestou je vždy alespoň jedna z kombinací z množiny:

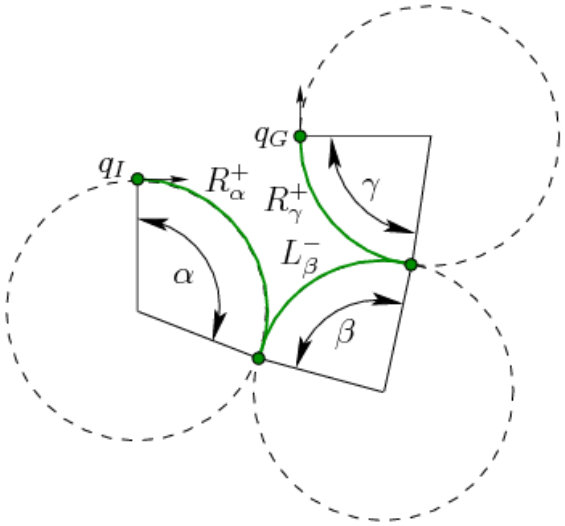
$\{C|C|C, CC|C, C|CC, CSC, CC_\beta|C_\beta C, C|C_\beta C_\beta|C, C|C_{\frac{\pi}{2}}SC, CSC_{\frac{\pi}{2}}|C, C|C_{\frac{\pi}{2}}SC_{\frac{\pi}{2}}|C\}$.

Pro implementaci bylo zvoleno použití Reeds – Shepp křivek, právě protože umožňují simulaci pohybu auta s Ackermanovým řízením a dovoluje mu přehazování rychlostí na zpátečku v průběhu jízdy. Oproti Grid Search najde přesnou cestu mezi konfiguracemi a je rychlejší pro větší vzdálenosti.

3.3.3 Grid Search

Tato technika plánování cesty definuje šest akcí: $L^+, L^-, R^+, R^-, S^+, S^-$. R označuje maximální zatočení vlevo, R doprava a S jízdu rovně. Index pak označuje směr jízdy (představuje znaménko rychlosti).

Plánovač udržuje strom T konfigurací a frontu $OPEN$ s ukazateli na konfigurace stromu, které zatím nebyly expandovány. Uzly v $OPEN$ jsou uloženy podle nákladu na cestu (Pro expanzi se vybírá nejdříve uzel s kratší cestou).



Obrázek 3.4: Příklad Reeds – Shepp křivky $R^+_\alpha L^-_\beta R^+_\gamma$, zjednodušeně $C|C|C$

Grid Search začíná inicializací počáteční konfigurace q_{start}, stromu T s kořenem q_{start} a přidáním q_{start} do fronty $OEPN$. Dokud je fronta $OPEN$ neprázdná nebo není dosažena maximální zadaná velikost stromu, opakuje se cyklus s expanzí stromu.

Expanze probíhá tak, že se z fronty $OPEN$ vybere jedna konfigurace q, zároveň je z fronty odstraněn. Jestli-že q je v blízkém okolí cíle q_{goal}, algoritmus končí úspěchem a vrací cestu stromem T z q_{start} do q. Pokud konfigurace q není v blízkém okolí jiné už expandované konfigurace (například v n-rozměrné mřížce, pro n-rozměrnou konfiguraci), provádí expanzi tak, že pro všechny akce $v \in \{L^+, L^-, R^+, R^-, S^+, S^-\}$ generuje novou konfiguraci q_{new}. Každou konfiguraci q_{new}, pro kterou je cesta z q do q_{new} bezkolizní, vloží uzel q_{new} do stromu T jako následovníka q a také do fronty $OPEN$.

Plánovač Grid Search při počítání ceny cesty bere v úvahu nejen vzdálenost, ale také změny směru jízdy a zatáčení. Na rozdíl od Reeds – Shepp křivek je nalezená cesta vždy nekolizní, kontrola kolizí je již zahrnuta do Grid Search.

Výhodou Grid Search je rovnoměrné hledání cesty a jeho vysoká úspěšnost. Pokud je velikost kroku dostatečně malá, metoda vždy najde cestu, jestli existuje. Plánování běží rychleji v omezeném prostoru, protože překážky ořezávají vygenerovaný strom.

Nevýhodou je, že plánovač nenajde přesnou cestu, ale jen aproximaci k cíli. Strom navíc roste polynomiálně v závislosti na uražené vzdálenosti.

3.4 Optimalizace cesty

Nalezená cesta pravděpodobnostním algoritmem bývá často zbytečně složitá a vyskytuje se v ní mnoho zbytečných uzlů navíc. Je zřejmé, že se s takovou cestou dá něco udělat. Optimalizace cesty je důležitou součástí hledání cest. [8]

Nalezenou cestu můžeme optimalizovat podle různých požadavků. Nejčastěji se jedná o počtu uzlů a celkové délce cesty, předmětem optimalizace může být i redukce velikostí změn směrů.

Nejčastější způsob optimalizace pracuje na principu odebírání uzlů z cesty a nahrazováním přímou hranou mezi předcházejícím a následujícím uzlem. Často je možné takto uzly spojit, a také se jen zřídka jedná o přeskočení jediného uzlu. Výběr dvou uzlů pro redukci tak může probíhat náhodně, nebo například tzv. hladovým přístupem.

Hladový přístup začíná kontrolovat nejdříve přímou cestu ze startu do cíle, pokud nelze spojit testuje startovní uzel s uzlem před cílem a dál postupně pokračuje dokud nedojde na uzel za startem. Pak to samé opakuje se začátkem posunutým o jeden uzel dál. Pro cestu s n uzly pak zkontroluje $(n-1)(n-2)/2$ dvojic.

Další způsob optimalizace vyhlazuje nalezenou cestu. Používá body nalezené cesty jako řídící body pro spline křivku, bezkolizní křivka pak nahradí celý úsek cesty.

Kapitola 4

Návrh appletu a webových stránek

Tato kapitola se zabývá návrhem appletu a webových stránek, které byly v průběhu práce realizovány.

4.1 Webové stránky

Implementační jazyk webových stránek bude HTML s definovaným stylem stránek pomocí CSS. Jazyk HTML je pro tyto účely nejvhodnějším řešením. Navíc je možné použití skriptovacího jazyku Javascript.

Stránky budou obsahovat:

- Úvod do problematiky.
- Stručný popis problémů neholonomních objektů.
- Stručný popis použitých algoritmů hledání cest.
- Aplikace demonstrující algoritmy hledání cest.
- Odkaz na pdf s prací.

Přímo na stránkách bude spustitelný applet demonstrující jednotlivé algoritmy (blíže k návrhu appletu níže). Webové stránky budou po dokončení zveřejněny na internetu spolu s dokumentací.

4.2 Ukázkový applet

Applet má být spustitelný jednoduše na webových stránkách, proto byl jako implementační jazyk zvolen Java. Tento programovací jazyk je velmi robustní a díky virtuálnímu stroji, na kterém běží, je také multiplatformní.

Applet bude vizualizovat problém hledání cest a jeho řešení. Je tedy nutné vyřešit vizuální složku. Pro vizualizaci byla zvolena knihovna java.awt pro svou jednoduchost, navíc umožňuje i zobrazení průhledných barev.

Applet má za úkol umožnit uživateli jednoduchou editaci scény, popřípadě načtení předdefinovaných scén. Uživatel musí být schopen měnit vlastnosti a tvar robota, nakonec spouštět všechny implementované algoritmy s různými parametry výpočtu. Proto bude applet rozdělen do tří módů.

První mód umožní editaci scény, tj. rozměry mapy, překážky uvnitř a polohu startovní a cílové pozice. Nebo také načítání předdefinovaných map.

Druhý mód umožní editaci vlastností, tj. jeho tvaru, a důležitých parametrů robota, jako je jeho poloměr rotace při maximálním otočení kol, nebo střed robota. Dále by měl umožnit načítání několika předdefinovaných vozidel pro snadnější rychlé použití.

Třetí mód pak následně uživateli umožní výběr algoritmu, nastavení parametrů výpočtu a následně řídit simulaci (musí umět simulaci zrychlovat, zpomalovat a krokovat).

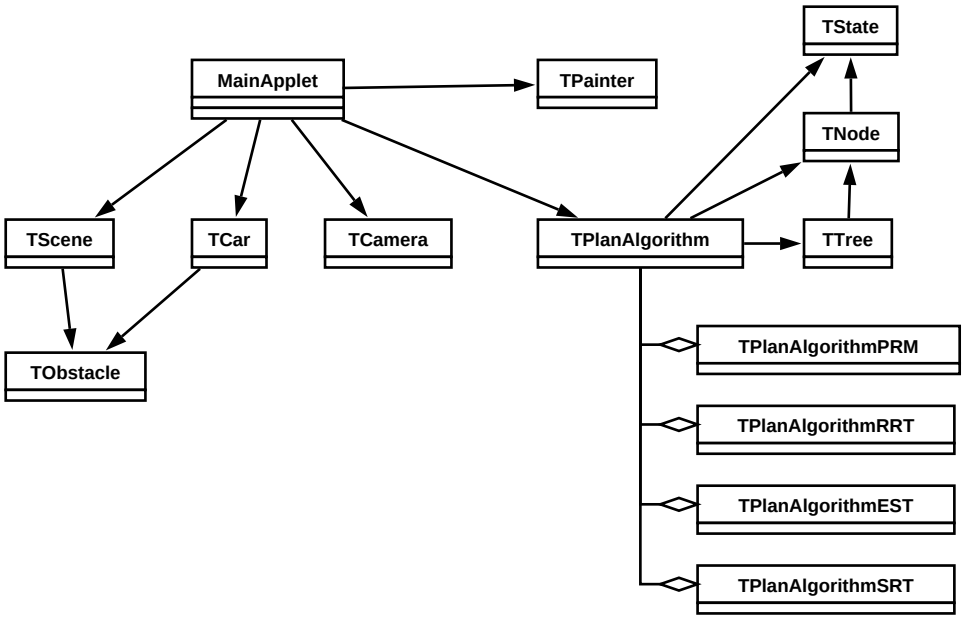
Applet bude podporovat vybrané algoritmy řešení úloh hledání cest neholonomních objektů. Uživatel bude mít možnost vizuálně nadefinovat vlastní mapu prostředí, ve které se bude pohybovat robot a tvar a vlastnosti robota (jako například přívěs, otáčivá a neotáčivá kola). Všechny objekty v prostředí (i robot) budou definovány jako polygon ve dvourozměrném prostředí.

Uživatel dostane možnost vybrat z implementovaných algoritmů a následně i nastavit parametry vyhledávání. Po spuštění simulace applet odsimuluje vyhledávání a následně vizualizuje výsledný pohyb robota po nalezené trajektorii.

Dále je zapotřebí návrh diagramu tříd a seznam tříd, které budou zapotřebí a jakou budou mít funkci. Zjednodušený navržený diagram tříd je zobrazen na obrázku 4.1.

Základní třídou je zde `MainApplet`, která bude obstarávat uživatelské rozhraní. Třída `TPainter` bude plnit vykreslovací úlohy. `TScene` uchovává všechny informace o dané scéně se seznamem překážek uvnitř, ty jsou typu `TObstacle`. `TObstacle` zároveň budou plnit funkci polygonů vozidla `TCar`. Pohyb kamerou pro získávání detailů nebo oddálených pohledů umožní třída `TCamera`.

Pro plánovací algoritmy byla navržena třída `TPlanAlgorithm`, bude jen obecně definovat plánování a potřebné parametry, díky tomu pak půjde jednoduše za běhu měnit potomky této třídy (konkrétní algoritmy) za jiné. Dále byly naplánovány pomocné třídy pro samotný výpočet. Jsou to `TState` definující stav vozidla, `TNode` obalující uzel grafu vytvářeného algoritmy a `TTree` zprostředkovávající stromové struktury grafu.



Obrázek 4.1: Zjednodušený návrh diagramu tříd appletu

Kapitola 5

Implementace

Tato práce se kromě teorie o pravděpodobnostních algoritmech a neholonomních objektech zabývá také vytvořením appletu, který problém hledání cest pro neholonomní objekty řeší a vizualizuje jej.

Tato kapitola se zabývá problémem implementace projektu. Applet byl implementován v jazyce Java, při práci s Javou byla používána převážně učebnice jazyka Java [5] a oficiální dokumentace Javy [2]. V následující části najdete bližší popis implementace projektu.

5.1 Popis tříd

V této části najdete popis všech použitých tříd v appletu, jejich popis a k čemu slouží. Můžeme je rozdělit do tří skupin podle jejich použití na řídící třídy, třídy popisující scénu a třídy, týkající se práce algoritmů.

5.1.1 Řídící třídy

Skupina řídících tříd má na starost řízení aplikace, inicializaci a komunikaci s uživatelem. Patří sem třídy `MainApplet`, `TPainter` a `TCamera`.

MainApplet

Tato třída je hlavní, reprezentuje celou aplikaci appletu a její funkcí je inicializovat všechny ostatní objekty a zprostředkovává uživatelské rozhraní.

Třída `MainApplet` dědí od třídy `Applet` nacházející se v základních balíčcích Java. Dále pak implementuje vlastnosti `MouseListener`, `MouseMotionListener`, `MouseWheelListener`, `KeyListener` a `Runnable`, umožňující řízení aplikace pomocí myši a klávesnice. Zároveň je `MainApplet` také třídou, která obsahuje statickou funkci `main`, kterou provádí v případě spuštění aplikace.

Po spuštění je volána funkce `init`, která inicializuje celou scénu, kameru, robota a doublebuffer (vytvoří kopii obrazu o velikosti okna appletu).

Následně applet běží v nekonečné smyčce, ve které obnovuje obraz v případě jakékoli změny (kreslí celou scénu do bufferu a ten pak celý vykreslí najednou pro odstranění nepříjemného blikání appletu). Pokud běží algoritmus, volá v pravidelných intervalech jeho funkci pro provedení kroku algoritmu.

TPainter

Třída pracuje jako sbírka statických funkcí určených pro vykreslování různých objektů do bufferu, k tomu používá třídu `Graphics2D`, nacházející se v knihovnách java.awt. Ta umožňuje kreslit několik základních tvarů, definovat šířku čar a některé jednoduché efekty, jako je čárkovaná nebo tečkovaná čára. V appletu je použita možnost kreslení průhlednými barvami (alpha channel).

TCamera

V celém appletu je jen jediná instance této třídy a její funkce jsou volány při každém vykreslování. Převádí totiž absolutní pozice vertexů polygonů na pozice v obraze v závislosti na pohledu, jehož informace jsou zde uloženy. Třída obsahuje pozici kamery a její přiblížení, na základě těchto hodnot provádí jednoduché dvourozměrné transformace. S kamerou pracuje přímo objekt `MainApplet` a umožňuje pohyb a přiblížení kamerou pomocí uživatelského rozhraní. Odkaz na `TCamera` je také předáván parametrem do všech vykreslovacích funkcí třídy `TPainter`.

5.1.2 Třídy pro popis scény

Druhá skupina popisuje scénu, její jednotlivé objekty a vlastnosti robota. Patří sem třída `TScene`, která popisuje scénu, `TCar` popisující auto a další pomocné třídy (viz níže).

TScene

Obsahuje v první řadě seznam všech překážek v podobě seznamu tříd `TObstacle` a zprostředkovává tvorbu a jejich editaci.

Tento seznam obsahuje navíc kromě překážek ve scéně tři další objekty, jsou to polygony znázorňující start, cíl a okraje mapy. Tyto objekty nelze smazat nebo jakkoli upravovat počet jejich vertexů. Zato je možné jednoduše aplikovat změny poloh vertexů stejným způsobem jako u ostatních objektů v seznamu. Dvoubodové polygony startu a cíle jsou definovány středovým bodem a směrovým bodem. Umožňuje tak uživateli nastavit jednoduše konkrétní konfiguraci robota. Třetí polygon značí okraje mapy, skládá se ze čtyř polygonů, které vždy tvoří obdélník.

TCar

Třída `TCar` definuje tvar a vlastnosti robota. Všechny polygony robota jsou uloženy obdobně jako ve scéně. Poloha a rotace robota spadá do jeho konfigurace v `TState` a nemá žádnou přímou vazbu s `TCar`. Třída `TCar` definuje pouze časově konstantní vlastnosti jako poloměr rotace při maximálním zatočení kol. Při vykreslování nebo výpočtech kolizí je potřeba všechny polygony nejdříve transformovat na absolutní pozice pomocí konkrétní konfigurace v `TState`.

TObstacle

Tento objekt znázorňuje polygony překážek a také polygony auta. Obsahuje jednoduchý seznam bodů polygonu a počet bodů. Jako překážka je uložen ve scéně s absolutními pozicemi v objektu `TScene`. Podruhé je objekt `TObstacle` použit u definice tvaru auta relativně vůči jeho středu a směru.

Při výpočtech kolize je nutné polygony robota transformovat na jeho absolutní pozice, to zajišťuje třída `TState`. `TObstacle` navíc obsahuje funkci pro detekci kolizí mezi dvěmi polygony, vrací seznam průniků hran polygonů, nekolidující polygony tak vracejí prázdný seznam.

TState

Tato třída je nejmenší třídou celé aplikace. Znázorňuje stav robota, neboli jednu konfiguraci. Uchovává data v podobě jednoho vektoru obsahujícího pozice a všechny rotace. Postupně jsou zde hodnoty po párech, první dvojice definuje polohu, druhá vektor ke středu otáčení, třetí vektor rotace vozidla, a následující jsou určeny pro otočení přívěsů. Kromě konstruktorů třídy lze volat metody pro transformaci polygonů auta na polygony s absolutní pozicí pro další použití jako výpočet kolizí nebo kreslení robota.

5.1.3 Třídy pravděpodobnostních algoritmů

Nakonec jsou v appletu třídy zajišťující provádění algoritmů a práci s grafy. Jsou to třídy `TNode`, `TTree` a `TAlgorithm`.

TNode

Důležitou třídou je `TNode`, představuje uzel grafu a uchovává přechody (hrany) mezi konfiguracemi. Každý uzel obsahuje odkaz na konfiguraci (`TState`), na strom, do kterého uzel patří (`TTree`) a všechny uzly, se kterými je spojen přechodem. Navíc také obsahuje odkaz na rodičovský uzel ve stromě pro snadné nalezení cesty ke kořenu stromu.

TTree

Třída `TTree` reprezentuje strom v grafu, obsahuje odkazy na všechny uzly stromu a odkaz na jeho kořen. Vlastní několik funkcí pro práci se stromy, jako je přidávání uzlů do stromu, nebo spojování stromů. Při hledání výsledné cesty pomáhá funkce pro přemístění kořene stromu do jiného uzlu. Pokud je hledána cesta stromem ze startu do cíle, pak stačí přesunout kořen do cíle a po rodičích se vždy nejkratší cestou dostaneme do cíle nejen ze startu, ale i z jakéhokoli jiného uzlu.

TPlanAlgorithm

Nakonec přichází ta nejsložitější třída. Jedná o jakousi sbírku základních metod a rozhraní pro implementované algoritmy, které z této třídy tyto metody dědí.

V prvé řadě třída obsahuje všechna možná nastavení pro různé algoritmy, jenž jsou nastavitelná pomocí uživatelského rozhraní. Důležitá nastavení jsou:

- Velikost a poloha okrajů mapy.
- Startovní a cílová konfigurace.
- Nastavení algoritmů (počet expanzí, počet generovaných stromů pro EST, počet pokusů o spojení stromů, délka kroku, atd.).

Třída `TPlanAlgorithm` obsahuje graf pro práci algoritmů v podobě seznamu uzlů (`TNode`), nepotřebuje zde zvlášť uchovávat hrany grafu, ty jsou uloženy již spolu s uzly v jediné třídě. Zato zachovává navíc seznam kořenů všech stromů obsažených v grafu pro pozdější použití.

Tato třída po inicializaci pracuje jako stavový automat, hlavní třída appletu pouze opakovaně posílá požadavky na další krok. `TPlanAlgorithm` počítá s fází, ve které se algoritmus konkrétně nachází, a počet provedených kroků jen obecně, konkrétnější použití připadá až na potomky této třídy.

Svým potomkům zanechává navíc sbírku užitečných metod:

- `clear` – vyčištění grafu a inicializace,
- `start` – inicializace a spuštění algoritmu,
- `step` – krok určen pro předefinování,
- `optimalizePath` – pokus o optimalizaci výsledné cesty,
- `addroot` – přidání uzlu jako nového stromu,
- `closest` – nejbližší uzel,
- `getDensity` – vrací hustotu rozložení uzlů (mřížka),
- `incDensity` – inkrementuje hustotu,
- `checkCollide` – Kontrola kolizí robota v konfiguraci,
- `stepexpandtree` – několik funkcí pro různé expanze (EST a RRT),
- `stepTryExpandNodeSpecAD` – pokus o expanzi s danou rotací a ujetou vzdáleností,
- `tryJoinRSNodes` – spojení dvou uzlů pomocí Reeds – Shepp křivky,
- `tryJoinNodes` – test zda je část křivky bezkolizní.

Bližší popis implementace jednotlivých pravděpodobnostních metod hledání cesty je v části níže 5.3.

5.2 Detaily imlementace

V následujícím textu je rozebrána implementace do menších detailů a problémů, které byly v průběhu implementace řešeny.

5.2.1 Model robota

Program simuluje robota, jehož konstrukce odpovídá autu s Ackermanovým řízením, což mu umožňuje pohyb vpřed i vzad. Otáčet se může pouze při jízdě a k tomu musí mít v patřičném směru natočená kola. Tvar auta, jeho kol a vozíku je definován jako množina libovolných polygonů, auto, kola i vozík tak mohou nabývat libovolných tvarů a velikostí.

Vozidlo má navíc definováno speciální parametry omezující a určující jeho pohyb. První parametr definuje minimální poloměr kružnice, po které auto může jet, neboli minimální poloměr rotace. Podle něj se při jízdě natáčejí kola na stranu.

Další tři parametry se týkají přívěsu. Jeden určuje vzdálenost bodu přípojení přívěsu od středu auta. Druhý určuje vzdálenost bodu přípojení přívěsu od středu přívěsu. A poslední definuje maximální možný náklon přívěsu oproti směru vozidla, kde již hrozí nebezpečí poškození.

5.2.2 Stav a transformace

Stav auta, tak jak je definován appletem, je dán jako vektor reálných proměnných typu `float`. Vždy po dvojicích určují dvourozměrný vektor pozice, normalizovaný vektor rotace, směrový vektor ke středu otáčení (pro Ackermanovo řízení kol) a směrový vektor přívěsu viz tabulka 5.1.

Pozice	Rotace	Otočení kol	Rotace vozíku

Tabulka 5.1: Konfigurace robora, každá položka je vektor o dvou reálných hodnotách.

Robot je definován pomocí listu polygonů, kde navíc každý polygon má přiřazenou informaci o typu polygonu podle vazby na robota. Polygon může být pevnou součástí robota nebo kolem řízeným pomocí Ackermanova řízení, nebo část přívěsu. Podle středu tohoto typu je určen způsob transformace polygonů.

Pevné části robota jsou transformovány podle polohy a rotace. Kola s Ackermanovým řízením jsou transformována stejně, navíc však otočena podle směrového vektoru ke středu otáčení, tak aby kola jízdou opisovala kružnici okolo stejného středu otáčení. Poslední typ přívěs je transformován stejně, jen následné otočení závisí na posledním vektoru, tím je směrový normalizovaný vektor přívěsu.

5.2.3 Detekce kolize

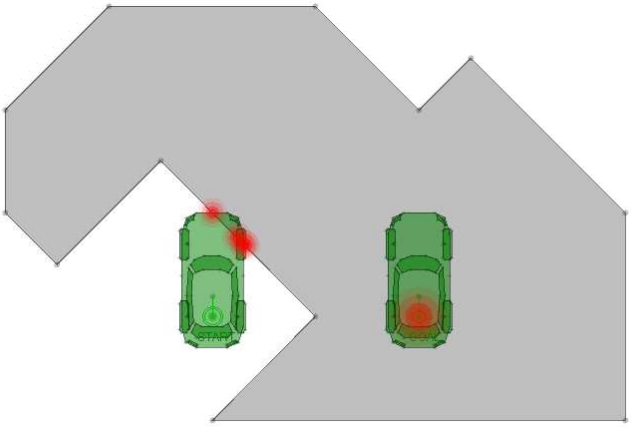
Detekce kolizí je základ celého plánování cesty. Robot musí dokázat odlišit cestu, která vede k nárazu do překážek od cest bezkolizních, jednoduše aby mohl vybrat tu bezkolizní.

Překážky i auto jsou v appletu definovány jako polygony se seznamy bodů a reálnými čísly typu float. Pro detekci kolizí v prvé řadě je použit výpočet průniků mezi hranami polygonů. Funkce, která vrací seznam průniků dvou polygonů, je definována ve třídě `TObstacle`. Kontrolují se všechny hrany jednoho polygonu ke všem hranám druhého polygonu.

Při testování výsledné aplikace nastal s tímto přístupem problém, a to že náhodně vygenerované pozice algoritmu PRM vkládali robota dovnitř překážek. Stačí však spočítat, kolik hran překážky se nachází přímo nad středem robota, je- li počet sudý, robot se nachází uvnitř překážky. Výsledek je ilustrován na obrázku 5.1. Další problém by mohl nastat při umístění malé překážky dovnitř vozidla, tento problém se však nevyskytne nijak často, plán by musel v prvé řadě obsahovat dostatečně malé překážky a pravděpodobnost zásahu vozidlem na překážku je velmi malá.

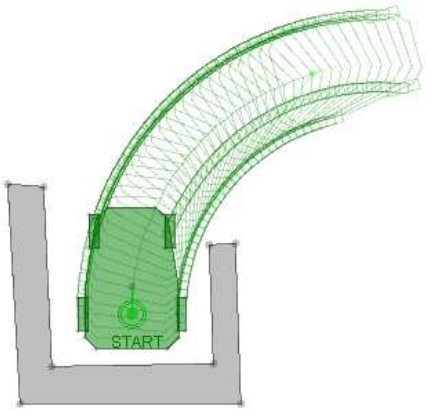
5.2.4 Přechody mezi konfiguracemi

Další důležitá část, kontrola kolizí přechodu mezi konfiguracemi, pracuje na základě půlení intervalu. Obě koncové konfigurace libovolného přechodu leží na jediné kružnici (nebo přímce), to umožňuje jednoduše určit polohu a směr robota uprostřed jeho pohybu na základě známého středu otáčení. Funkce `tryJoinNodes` je volána rekurzivně do zadané hloubky pro obě nově vzniklé poloviny a kontroluje kolizi středové konfigurace. Jakmile



Obrázek 5.1: Detekce kolizí (průniky polygonů s překážkou vlevo, auto schované uvnitř překážky vpravo)

jediná funkce vrátí kolizi, vrací i kořenová funkce kolizi. Testování kolizí přechodu je ilustrováno na obrázku 5.2.



Obrázek 5.2: Detekce kolizí přechodu mezi konfiguracemi pomocí půlení intervalu na kružnici.

Funkce pro kontrolu kolizí mezi konfiguracemi se nachází v objektu `TPlanAlgorithm`, a je tedy přístupná všem potomkům. Použití nachází při konstrukci Reeds – Shepp křivek a expanzi uzlů.

5.2.5 Expanze uzlů

Pro expanzi uzlů byly naprogramovány hned dva druhy expanze uzlů, jsou to expanze uzlů RRT a EST. Obě varianty jsou popsány níže v nadcházející části.

Expanze EST

Expanze EST vybírá náhodný uzel s náhodným rozložením. Pro definici náhodného rozložení slouží mřížka hustoty, která obsahuje implicitně $20 \times 20 \times 8$. První dvě hodnoty rozdělují prostor po ose x a y, třetí hodnota dělí rotaci vozidla na osm segmentů, čímž zajišťuje možnost expanze do míst s novým směrem, i když je místo plné v ostatních směrech. Celá mřížka je definována jako pole typu `int`. Při každém vytvoření nějakého uzlu je příslušná hodnota inkrementována o 1. Při zpětném zjištění hustoty stačí sáhnout do paměti a není potřeba opakovaných výpočtů hustoty.

Po výběru uzlu jsou postupně generovány náhodné konfigurace, do kterých se robot může dostat po jediné kružnici s maximální vzdáleností uživatelem určenou omezující podmínkou. První pokus je přijat jen v případě, že je hustota v buňce vygenerované konfigurace rovna nule a není kolizní sám ani přechod k ní. Je-li uzel nepřijat, opakovaně se generují nové uzly a zvyšuje minimální hranice přijetí.

V porovnání s expanzí bez kontroly hustoty tento přístup nevytváří extrémní shluk konfigurací při kořenech stromů. Vede k rovnoměrnějšímu rozložení a umožňuje algoritmu dostat se do větších vzdáleností s nižším počtem uzlů.

Expanze RRT

Princip RRT generuje náhodné body po prostoru a hledá nejbližší existující uzel expandovaného stromu. Ten je následně expandován po kružnici, tak aby se přiblížil vygenerovanému bodu. Maximální vzdálenost expandovaného uzlu je omezena hodnotou `stepsize` zadanou uživatelem. Není-li nový uzel ani přechod mezi nejbližším a novým uzlem proveditelný, generuje se další náhodný bod, jinak je nový uzel přijat do stromu.

Tento způsob je jednoduchý a často má výborné výsledky, prochází prostor rovnoměrně i v malých počtech uzlů.

5.2.6 Hledání reeds – shepp křivky

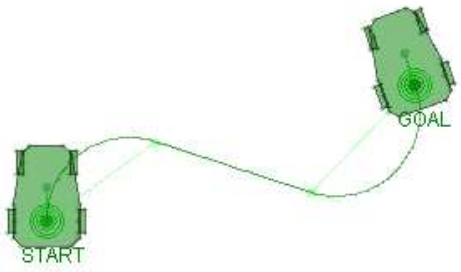
Hledání reeds – shepp křivek bylo pro projekt asi nejdůležitějším krokem. jejich síla spočívá v přípustném a jednoduchém spojení mezi dvěma libovolnými konfiguracemi. Použití v projektu nachází při spojování stromů.

Implementace byla provedena podle definice v sekci 3.3.2. Tedy výsledná křivka mezi konfiguracemi se skládá z několika úseků, z nichž každý úsek je jeden přechod (každý přechod z definice appletu musí být úsečka, nebo část kružnice) mezi dvěma konfiguracemi. Proto jsou do grafu při každém platném spojení přidány minimálně dvě další konfigurace (nové dva uzly).

Nechť x_1 a x_2 jsou konfigurace, mezi kterými je hledána reeds – shepp křivka. První a poslední segment křivky bude vždy zatáčení. Tedy na začátku jsou umístěny dvě kružnice se středy v bodech c_1 a c_2 tak, že x_1 leží na kružnici kolem c_1 a x_2 leží na kružnici kolem c_2 a jejich směrové vektory tvoří tečny těchto kružnic. Obě kružnice mají poloměr rovný minimálnímu poloměru otáčení robota. Pak body c_1 a c_2 mají jen dvě možná umístění podle toho, kam vozidlo má zatáčet v daných segmentech. c_1 je tedy na kolmici od x_1 vpravo, pokud je nutné z x_1 do x_2 zatáčet doprava, to určí znaménko skalárního součinu vektorů normály směrového vektoru x_1 a rozdílu pozičních vektorů $x_1 - x_2$. To samé platí i pro c_2. Následně je položena jediná tečna obou kružnic, která spojí obě kružnice, musí však zachovat orientaci vozidla, tj. pokud vozidlo opouští po předu první kružnici, musí vozidlo po předu vstupovat do druhé kružnice. Tečna je zvolena tak, aby brala kratší verzi

ze dvou zbývajících možností. Vzniklé dotykové body tečny s kružnicemi jsou následně spojeny hranou mezi sebou a k uzlům x_1 a x_2, vzniká tak cesta mezi uzly x_1 a x_2. Pokud jsou všechny tři přechody proveditelné, oba nové body jsou umístěny do grafu a stromy tím spojeny.

Výsledná reeds – shepp křivka spadá do množiny CSC. Její výhodou je vysoký dosah, protože člen S uprostřed není omezen poloměrem rotace. Křivka tak může bez problému spojovat i stromy diferenciálně řízeného robota. Příklad reeds – shepp křivky je na obrázku 5.3.



Obrázek 5.3: Příklad vygenerované reeds – shepp křivky mezi startem a cílem

5.2.7 Model přívěsu

Přívěs je definován množinou polygonů s relativní pozicí vzhledem k vozidlu a třemi parametry. Parametry jsou `hook` (určuje místo napojení vozíku k autu, nebo také osu otáčení vozíku), `center` (střed vozíku neboli bod, ve kterém je přívěsem možno pohybovat pouze dopředu a dozadu) a `maxangle` (určuje maximální otočení přívěsu).

Při expanzi stromů je jízda s vozíkem simulována diferenciální rovnicí $\alpha' = \frac{w}{d} \sin\gamma$, kde α je úhel otočení přívěsu oproti vozidlu, α' je derivace úhlu přívěsu neboli jeho úhlová rychlost, w je rychlost, jakou je přívěs tažen, d je vzdálenost středu přívěsu od ukotvení (`center`), γ je úhel tažení přívěsu. Přívěs nemusí být ukotven ve středu vozidla, proto w není rovno rychlosti vozidla, ale je o něco větší. dále platí $\gamma = \alpha + \theta$, kde θ značí odchýlení tažení vozíku způsobené polohou úchytu a zatáčením vozidla. θ a w jsou v celém přechodu konstantní (celý přechod leží na jediné kružnici). Zjednodušeně bez konstant je tedy nutné řešit diferenciální rovnici $\alpha' = \sin\alpha$. Řešení rovnice bylo implementováno pomocí metody Runge – Kutta čtvrtého řádu.

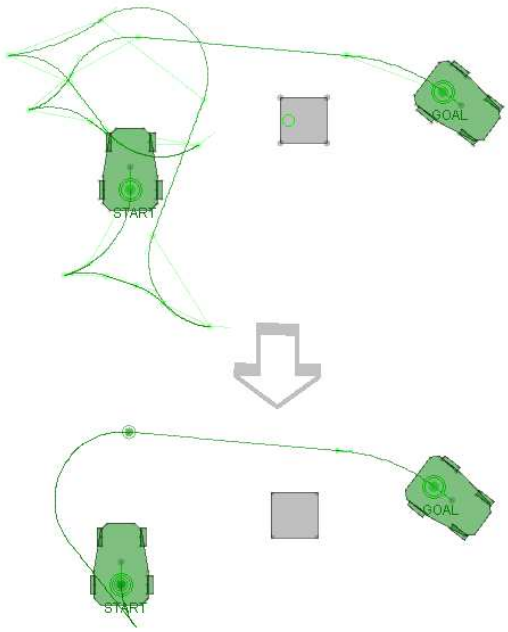
Pro kontrolu validity stavů přibyla nová podmínka, která kontroluje, zdali není vozík otočen více než udává parametr `maxangle`. Výpočet přívěsu ovšem probíhá, pouze když aktivní robot vlastní nějaký polygon typu přívěs.

Při použití již implementovaných reeds – shepp křivek však vznikl problém. Po aplikaci křivky sice robot je na správné pozici a se správným natočením, ale přívěs nepasuje. Proto byla navrhnuta další křivka, která už nepatří do množiny možných optimálních křivek. Jedná se o klasický pohyb auta, který řidič provede aby auto bylo znovu na stejném místě, ale přívěs byl otočen. Křivka má ve značení reeds – shepp křivek tvar $C_\beta C_{-2\beta} C_\beta | S$. Při spojování stromů se tedy vygeneruje křivka pro neholonomního robota bez přívěsu, následně potom křivka pro upravení polohy přívěsu.

5.2.8 Optimalizace cesty

Výsledky pravděpodobnostních algoritmů jsou často hrubé a matoucí, často vyjde klikatá cesta i přesto, že je možné jet rovně. Proto se po nalezení cesty používá optimalizace.

Po nalezení cesty zadaným algoritmem je spuštěna optimalizace výsledné cesty. Celá cesta je prohledávána od největších částí cesty po menší a každá z těchto částí je testována, zda-li ji není možné nahradit reeds – shepp křivkou. Optimalizace se zastaví, až když není možné žádnou reeds – shepp křivkou zmenšit počet uzlů cesty (viz obrázek 5.4).



Obrázek 5.4: Ukázka optimalizace cesty

Výsledkem je cesta s minimalizovaným počtem uzlů. Další možná optimalizace by mohla zkracovat skutečnou délku cesty nebo snižovat prudkost zatáček.

5.3 Implementované algoritmy

Děděním vlastností třídy `TPlanAlgorithm` vzniklo několik jejich potomků, které mají na starosti konkrétní algoritmy upravené pro neholonomní objekty. Implementovány byly algoritmy PRM, RRT, EST a SRT.

Důležitými nastavitelnými parametry algoritmů jsou `roots` – počet náhodně vygenerovaných kořenů, `expands` – počet expanzí stromů, `joints` – počet pokusů pro spojení stromů nebo uzlů a `stepsize` – maximální vzdálenost expandovaných uzlů. Všechny tyto parametry jsou jednoduše nastavitelné uživatelem v prostředí aplikace.

5.3.1 PRM

Implementován jako první byl algoritmus PRM. Pracuje jako stavový automat, každý stav představuje jednu fázi algoritmu. Fáze v pořadí jsou: inicializace, generování náhodných

uzlů, spojování uzlů do grafu, přidání a napojení startu a cíle, nalezení cesty grafem a následná optimalizace. Pracuje pouze s dvěma parametry `roots` a `joints`. Průběh implementovaného algoritmu vypadá následovně:

1. Inicializace.
2. Přidávání náhodných uzlů do grafu (celkem `roots` uzlů).
3. Spojování uzlů do grafu (každý `joints` pokusů s ostatními pomocí reeds – shepp).
4. Přidání a napojení startu a cíle.
5. Nalezení cesty grafem (nenajde-li cestu algoritmus končí).
6. Optimalizace cesty.
7. Vizualizace cesty.

Každý generovaný uzel představuje jednu konfiguraci robota, tj. nejen pozice, ale i jeho rotace. Pro spojení dvou náhodných konfigurací je zapotřebí použití reeds – shepp křivek. Stejným způsobem je napojen start a cíl. Výsledný graf je les, hledání cesty je tedy velmi jednoduché. Kořen stromu, ve kterém se nachází cíl, je přesunut do cíle a pak cesta vznikne sledováním rodičů ze startovní konfigurace. Touto technikou vzniká spíše chaotická cesta, ta je však záhy optimalizována.

5.3.2 RRT

Druhý implementovaný algoritmus je RRT. Stejně jako PRM je dědicem třídy `TPlanAlgorithm`. používá však parametry `expands`, `joints` a `stepsize`. Používá RRT expanzi v sekci 5.2.5. Průběh implementovaného algoritmu RRT vypadá následovně:

1. Inicializace.
2. Přidávání startu a cíle do grafu.
3. Expanze RRT (celkem `expands` pokusů o expanzi do vzdálenosti `stepsize`) a test na spojení pomocí reeds – shepp v nově vygenerovaných uzlech, jsou-li spojeny, algoritmus dál neexpanduje.
4. Nalezení cesty grafem (nenajde-li cestu algoritmus končí).
5. Optimalizace cesty.
6. Vizualizace cesty.

Tento algoritmus se pokouší expandovat oba stromy najednou ve směru k náhodné souřadnici. Pokud se novými uzly stromy dostanou dostatečně blízko, že se je podaří spojit reeds – shepp křivkou, algoritmus končí expandování, nalezení cesty je už zcela jasné.

5.3.3 EST

Dalším algoritmem, jenž byl v rámci appletu implementován, je EST. Stejně jako RRT používá parametry `expands`, `joints` a `stepsize`. Používá EST expanzi definovanou v sekci 5.2.5. Průběh jednotlivých fází EST je následovný:

1. Inicializace.
2. Přidávání startu a cíle do grafu.
3. Expanze EST (celkem `expands` pokusů o expanzi do vzdálenosti `stepsize`).
4. Spojování stromů (`joints` pokusů o spojení pomocí reeds – shepp).
5. Nalezení cesty grafem (nenajde-li cestu algoritmus končí).
6. Optimalizace cesty.
7. Vizualizace cesty.

Implementovaný EST expanzi postupně střídá mezi oběma stromy. Každá expanze je ve vzdálenosti nula až `stepsize` a s náhodnou rotací oproti původnímu uzlu. Spojování stromů probíhá až na konec expandování.

5.3.4 SRT

Poslední algoritmus byl implementován hned ve dvou variantách. Jeho definice dovoluje použít jak RRT expanzi, tak EST expanzi stromů. Vytvořeny tedy byly hned dvě třídy. Jako jediný algoritmus používá všechny čtyři parametry. Průběh jednotlivých fází SRT pro obě varianty je následovný:

1. Inicializace.
2. Přidávání startu a cíle do grafu.
3. Přidávání náhodných kořenů do grafu (celkem `roots` uzlů).
4. Expanze RRT/EST (celkem `expands` pokusů o expanzi do vzdálenosti `stepsize`).
5. Spojování stromů (každý strom `joints` pokusů o spojení pomocí reeds – shepp s ostatními stromy).
6. Nalezení cesty grafem (nenajde-li cestu algoritmus končí).
7. Optimalizace cesty.
8. Vizualizace cesty.

Algoritmus SRT začíná jako PRM generováním náhodných konfigurací, ty však nejsou hned spojovány, ale použijí se jako základ pro seznam kořenů, ze kterých bude následovně expandován strom. Vzniklé stromy jsou postupně spojovány do jediného grafu.

Kapitola 6

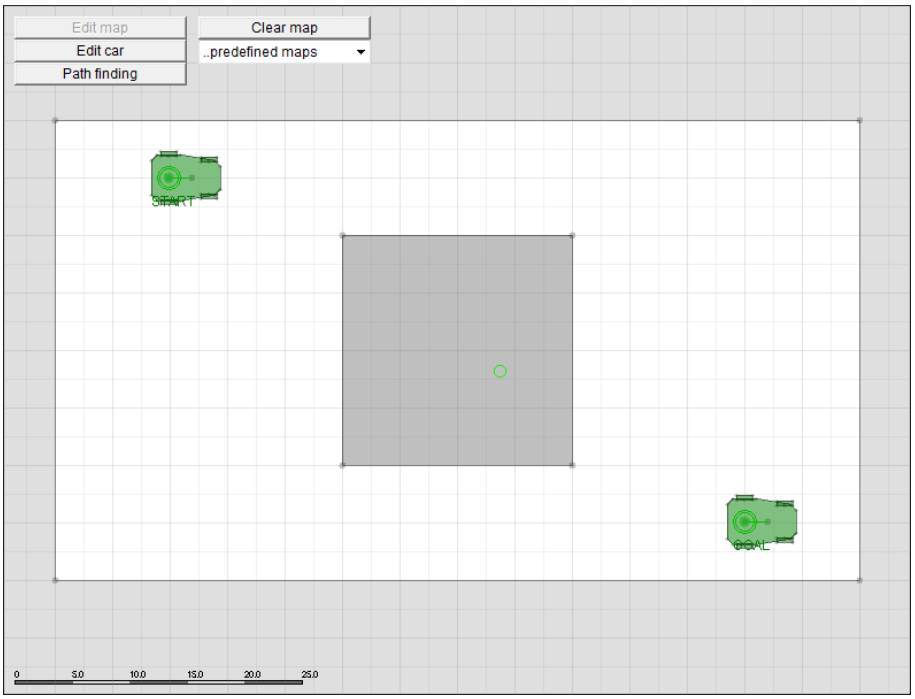
Popis appletu

Applet byl implementován a implementace byla popsána v předcházející kapitole. Tato část popisuje jak applet vypadá a jak jej ovládat ze strany uživatele.

6.1 Prostředí a ovládání appletu

Práce s appletem má čtyři hlavní části, které je třeba popsat pro snadné ovládání. První část je ovládání pohledu, druhá část je editor scény, třetí částí je editor robota a konečně plánovaní cesty.

Start appletu začíná v módu editace scény. První scéna, se kterou je applet spuštěn, je velmi jednoduchá, obsahuje start a cíl s jedinou čtvercovou překážkou uprostřed (viz obrázek 6.1).



Obrázek 6.1: Appletu po startu

Applet poskytuje uživateli mnoho možností editace a nastavení algoritmů. V následující části je popsáno vše, co uživatel může s touto aplikací provést.

6.1.1 Základní ovládání

Zaprvé je zde myší ovládaný pohled. Pohled se posune tahem pravým nebo levým tlačítkem (pozor, levým stiskem na vertex potáhne vertex, doporučuji posouvat pohled pravým tlačítkem) a je přiblížen/oddálen otočením kolečka na myši nebo tahem se stisknutým kolečkem, a nebo tlačítky „Q“ a „Z“. Klávesa „G“ zapíná/vypíná zobrazení mřížky pro zachytávání bodů. To je vše k pohledu.

Vlevo nahoře jsou tlačítka umožňující přechod mezi módy appletu, kdykoli je možné přepnout applet do jednoho ze tří módů. Jsou to „Editace scény“, „Editace robota“ a „Plánování“. Postupně jsou všechny níže rozepsány.

6.1.2 Editace scény

Tato část obsahuje tvorbu a modifikaci překážek. První překážku vytvoříte kliknutím do prázdného prostoru, vertexy do označené překážky pak přidáte stejným způsobem. Vertexy je možné posouvat tahem levým tlačítkem. Označit vertex a tím i překážku je možné kliknutím levého tlačítka na daný vertex. Klikem pravého tlačítka zrušíte označení, dalším kliknutím na nějaký vertex jej smažete.

Při tvorbě překážek si určitě všimnete, že body přidáváte v jednom směru od označeného vertexu, směr jde obrátit kliknutím kolečka na myši.

Následují ještě některé funkční klávesy. Stisk „M“, „R“, „S“ způsobuje posun, rotaci a zvětšení celého polygonu. Stisk „D“, „X“ způsobuje kopírování a mazání objektu (viz přehled kláves v tabulce 6.1).

Start a cíl jsou také polygony, pracuje se s nimi podobně, jen na ně nefungují funkce kláves a nejde manipulovat počet vertexů. Start i cíl se skládá ze dvou bodů, středový a směrový vertex. Posunem středu posunete celého robota, posunem směrového vertexu měníte směr robota. Dále je možné obdobně měnit i obdélníkové hranice scény.

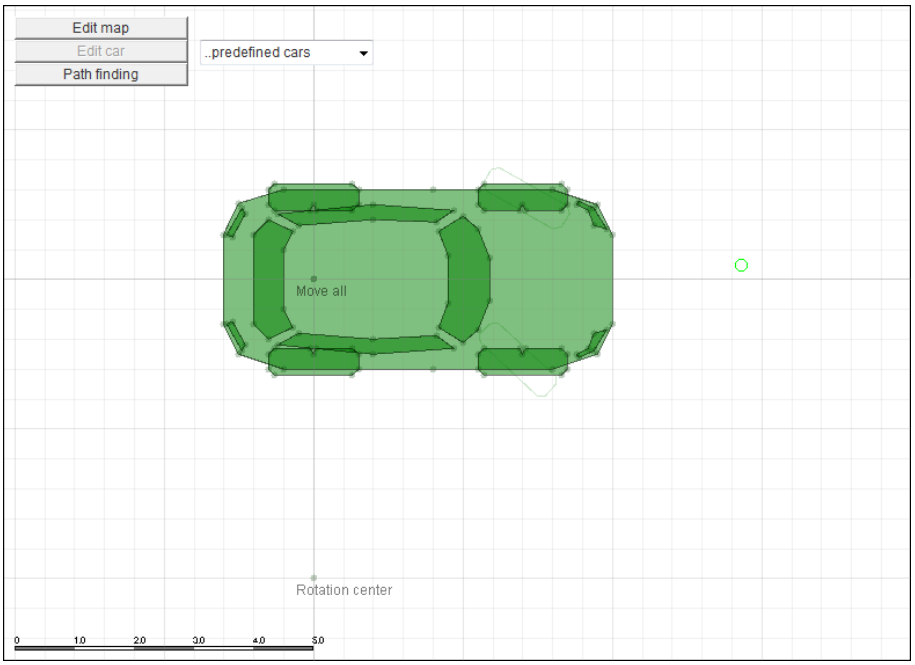
Pro rychlou ukázku slouží načítání předdefinovaných scén. Vlevo nahoře vedle volby módů se nachází komponenta obsahující seznam scén, které si uživatel může načíst. Je viditelná pouze v módu editace scény.

Klávesa	**Funkce**
Q	Přiblížení pohledu
Z	Oddálení pohledu
G	Přepnutí mřížky
M	Posun celé překážky
R	Rotace překážky
S	Zvětšení překážky
X	Smazání překážky
D	Kopie překážky

Tabulka 6.1: Přehled použitých kláves v módu editace scény.

6.1.3 Editace robota

Do této části se dostanete kliknutím na tlačítko „Edit car“. V tomto módu pracujete přímo s polygony robota. Kromě nich je zde i několik řídících vertexů, kterými je možno modifikovat vlastnosti robota. Vertex s názvem „Rotation center“ udává osu maximální rotace robota, vertex „Move all“ je v podstatě místo, kde kolmice osy robota protíná střed rotace (střed robota), a posunutím tohoto bodu posunete celého robota.



Obrázek 6.2: Editace robota

Polygony robota jsou duplikovány na druhou stranu osy robota, tím tvoří buď dva různé polygony nebo jeden spojený. Tato funkce jde přepínat nebo i vypnout pomocí klávesy „Y“. Klávesa „T“ přepíná typ polygonu, k dispozici je pevná neotočná část a otočné kolo s Ackermanovým řízením (jeden z vertexů je střed otáčení kola označen „wheel axis“). Aktuální typ právě označeného polygonu je napsán vlevo nahoře pod volbou módu. Seznam všech kláves je popsán v tabulce 6.2.

Pro rychlou ukázku je zde také jedno urychlení. Načítání předdefinovaných vozidel se nachází vlevo nahoře vedle volby módů (pouze v módu editace robota). Obsahuje seznam několika vozidel, které si uživatel může načíst.

6.1.4 Plánování

Poslední zbývající mód umožňuje nastavení a spouštění výpočtu cesty. Poté co uživatel dle libosti nastaví scénu a robota může přejít k hledání cesty.

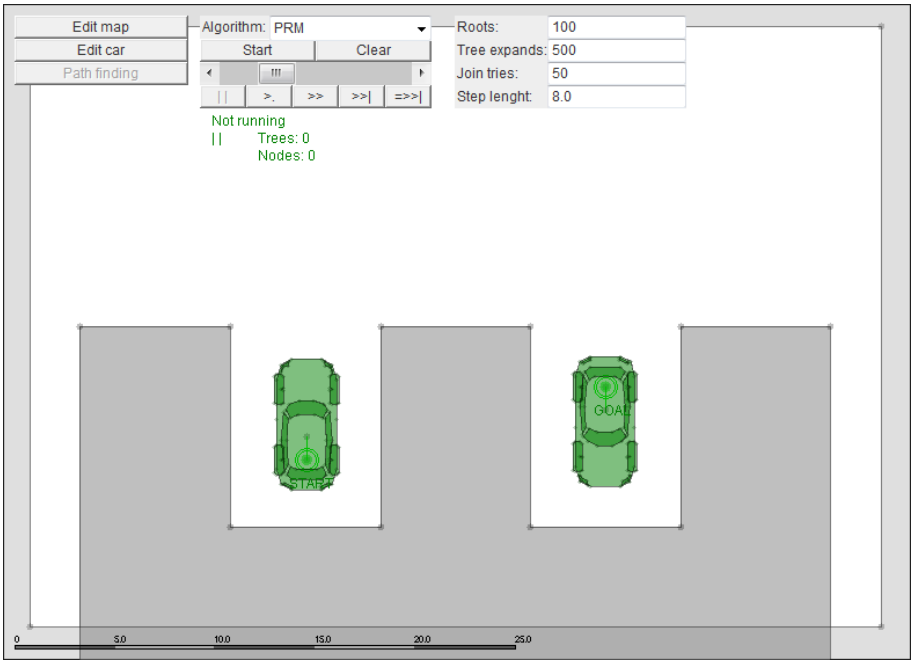
Po přepnutí do tohoto módu se objeví více nastavení než v ostatních módech (obrázek 6.3). Zaprvé vlevo nahoře se nachází výběr algoritmu, podle kterého se hledání cesty řídí. V pravém sloupci jsou nastavitelné parametry algoritmu. Parametr „Roots“ definuje počet náhodně generovaných uzlů ve volném konfiguračním prostoru (PRM a jako kořeny SRT). Parametr „Tree expands“ určuje počet expanzí stromů (RRT, EST a SRT). Následuje pa-

Klávesa	Funkce
Q	Přiblížení pohledu
Z	Oddálení pohledu
G	Přepnutí mřížky
M	Posun části
R	Rotace části
S	Zvětšení části
X	Smazání části
D	Kopie části
Y	Změna symetrie části
T	Změna typu části (pevná, kolo, přívěs)

Tabulka 6.2: Přehled použitých kláves v módu editace scény.

rametr „Join tries“, omezující počet pokusů o spojení stromů nebo uzlů. Poslední parametr „Step lenght“ omezuje maximální vzdálenost expandovaného uzlu.

Po nastavení parametrů uživatel může spustit výpočet kliknutím na tlačítko „start“ a pustit simulaci pomocí „>>“, nebo krokovat „>.“, rychlost běhu výpočtu závisí na poloze scrollbaru pod tlačítkem start. Pro zastavení slouží „| |“. Výpočet se zastaví na konci fáze stiskem „>>|“, a pro rychlý skok na další fázi slouží „=>>|“ (viz tabulka 6.3).



Obrázek 6.3: Plánování cesty robota.

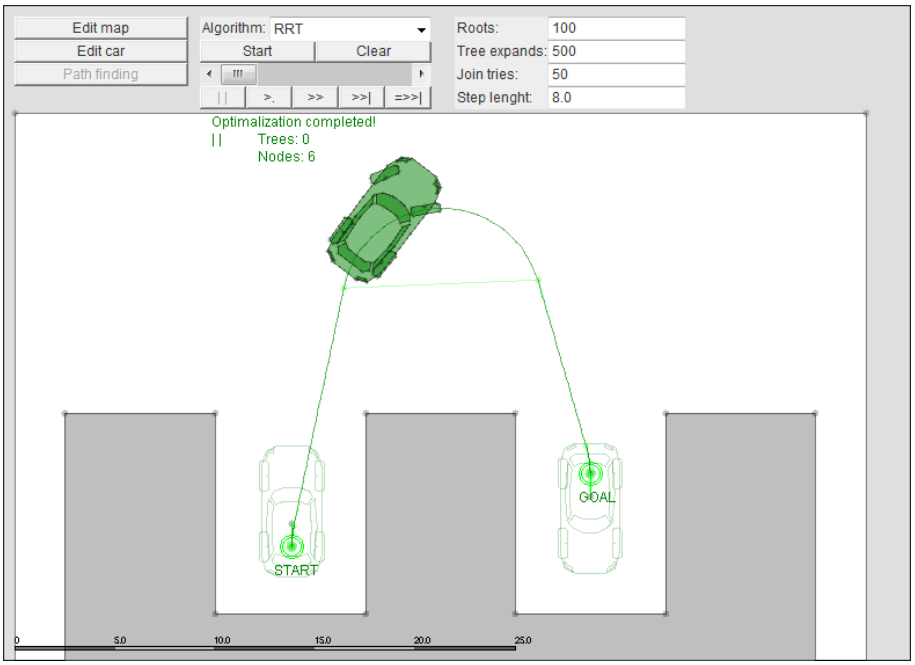
Průběžný stav výpočtu, tj. fáze, procentuální vyjádření pokroku ve fázi, celkový počet uzlů a zatím oddělených stromů je umístěn pod levým panelem s ovládáním běhu výpočtu.

Po dokončení všech fází výpočtu cesty přichází vizualizace pohybu robota po nalezené cestě (obrázek 6.4. Rychlost jeho pohybu je určena scrollbarem, tím je možné jej zastavit na místě.

Tlačítko	Funkce
Start	Inicializace a start výpočtu
Clear	vyčištění a zastavení výpočtu
| |	Pauza
>.	Krok
>>	Vpřed
>>|	Vpřed se zarážkou na konci fáze
=>>|	skok na další fázi

Tabulka 6.3: Tlačítka pro ovládání výpočtu.

Výpočet je tedy hotov, ať už algoritmus cestu najde nebo nenajde, uživatel může znovu upravit scénu nebo parametry dle libosti a znovu algoritmus opakovaně spustit.



Obrázek 6.4: Vizualizace cesty robota.

Kapitola 7

Testování

Poté, co byl applet a webové strány implementovány, bylo zapotřebí provést testování. Webové stránky byly testovány na prohlížečích Mozilla Firefox, Opera, Google Chrome a Internet Exploler. V průběhu implementace appletu bylo provedeno nespočet testů funkčnosti s různými parametry a s různými kritickými situacemi. Tyto testy však nebyly průběžně řádně zdokumentovány. Proto posléze vzniklo několik testů výsledného appletu, které prověří úspěšnost jeho implementace.

Pro účely testování vzniklo několik předdefinovaných map s překážkami a sada předdefinovaných vozidel na testy. Mezi vozidly jsou jak diferenciální vozidla, tak i vozidla s Ackermanovým řízením.

7.1 Test úspěšnosti algoritmů

Všechny algoritmy byly podrobeny zkouškám úspěšnosti na různých scénách a výsledky byly zapsány do tabulek. Testy tak pomohou stanovit parametry a algoritmy, které vedou k rychlému a efektivnímu řešení různých problémů.

7.1.1 Test na scéně „One box“

Jedná se o jednoduchou scénu, která se objeví hned jako první po startu aplikace (obrázek 6.1). Testovány byly postupně všechny algoritmy s různými parametry. Zaznamenána byla úspěšnost nalezení cesty z deseti pokusů běhu. Výsledky testu jsou v tabulce 7.1.

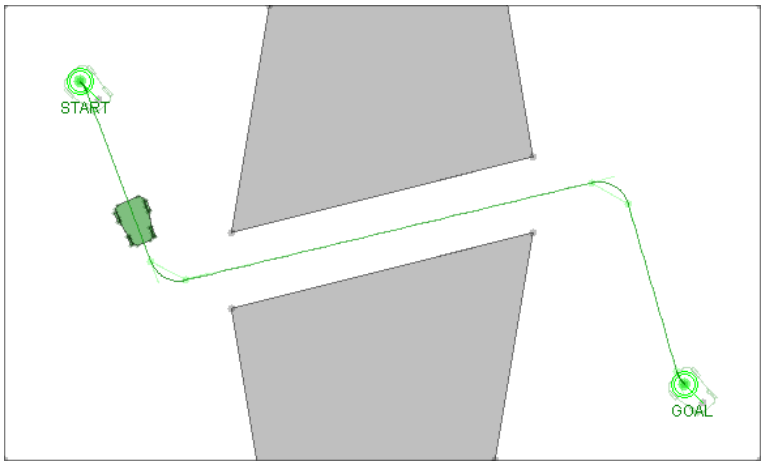
Algoritmus	Roots	Expands	Joints	Step lenght	úspěšnost
PRM	30	-	50	-	100%
RRT	-	50	50	8.0	100%
EST	-	100	50	15.0	90%
SRT (RRT expanze)	10	50	30	8.0	100%
SRT (EST expanze)	10	50	30	15.0	100%

Tabulka 7.1: Výsledky testu 1.

Všechny algoritmy s touto scénou i na nízká nastavení parametrů neměly prakticky žádný problém, pouze algoritmus EST se nedostal s 50 uzly za překážku (nejdříve zaplňoval prostor kolem startovní a cílové pozice), proto bylo třeba zvýšit počet expanzí na 100. Vyšší nastavení pravděpodobnost nalezení cesty zvyšuje a při menších pak záleží až příliš na náhodném faktoru, proto přejdeme na jinou scénu.

7.1.2 Test na scéně „Narrow“

Tato scéna obsahuje dvě překážky, mezi kterými se vozidlo musí protáhnout úzkým prostorem (viz obrázek 7.1). Použito bylo implicitně nastavené vozidlo s Ackermanovým řízením. Testovány byly znovu všechny algoritmy a jejich úspěšnost byla zaznamenána do tabulky 7.2.



Obrázek 7.1: Příklad nalezené cesty ve scéně „Narrow“

Algoritmus	Roots	Expands	Joints	Step lenght	úspěšnost
PRM	30	-	50	-	60%
PRM	60	-	50	-	90%
RRT	-	50	50	8.0	40%
RRT	-	100	50	8.0	70%
RRT	-	200	50	8.0	80%
EST	-	100	50	15.0	10%
EST	-	200	100	15.0	20%
EST	-	400	400	15.0	80%
SRT (RRT expanze)	20	100	30	8.0	90%
SRT (EST expanze)	20	100	30	15.0	80%

Tabulka 7.2: Výsledky testu 2.

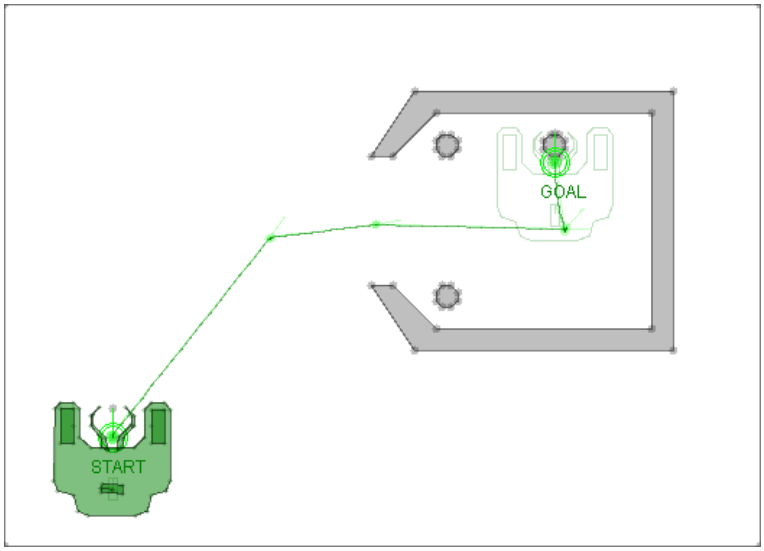
Výsledky testů ukázaly velkou výhodu náhodně generovaných uzlů z volného konfiguračního prostoru. Jak PRM, tak i SRT má s menšími počty uzlů větší úspěšnost na této úloze. EST má pořád problém dostat se na větší vzdálenosti, místo toho zaplňuje svou mřížku hustoty dalšími uzly, probírá tak podrobněji menší prostor.

7.1.3 Test na scéně „Trilobot Quest“

Tato scéna byla inspirována robotem Trilobot, jedná se o malého robota s diferenciálním řízením, stojí na dvou kolech s krokovými motory a jedním podpůrným kolem vzadu. Trilobot určený pro výukové účely je osázen množstvím různých senzorů.

Jedním z úkolů, které mu byly předkládány, bylo sebrat plechovku ze země robotickými kleštěmi, jenž má v přední části. Scéna byla obohacena o ohrádku ve které se nachází osmi-

úhelník představující plechovku. Trilobot má za úkol vjet do ohrádky a najet do pozice, kde je možné plechovku uchytit (viz obrázek 7.2). Testovány byly postupně všechny algoritmy a jejich úspěšnost byla zaznamenána do tabulky 7.3.



Obrázek 7.2: Příklad nalezené cesty ve scéně „Trilobot Quest“

Algoritmus	Roots	Expands	Joints	Step lenght	úspěšnost
PRM	30	-	50	-	0%
PRM	60	-	50	-	10%
PRM	120	-	100	-	40%
PRM	200	-	100	-	50%
RRT	-	50	50	4.0	50%
RRT	-	100	50	4.0	80%
RRT	-	200	50	4.0	100%
EST	-	70	100	8.0	90%
EST	-	100	100	8.0	100%
SRT (RRT expanze)	20	100	30	2.0	70%
SRT (EST expanze)	20	100	30	4.0	80%

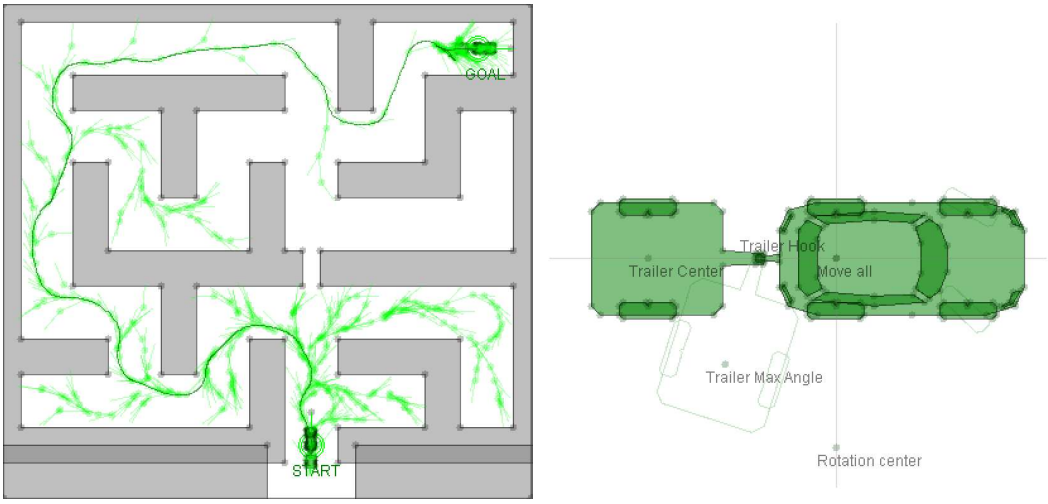
Tabulka 7.3: Výsledky testu 2.

Z výsledků je patrné, že algoritmus EST vynikl na tomto problému před ostatními. EST dokázal uvnitř ohrádky efektivně manévrovat na malém prostoru a jakmile jej zaplnil, pravděpodobnostní funkce na bázi hustoty v mřížce jej donutila expandovat mimo ohrádku, spojení stromů je pak už jednoduché. Algoritmy RRT a SRT si vedly také dobře. Zato PRM naprosto zklamal, jeho náhodně generované uzly jen málokdy zasáhly místo uvnitř ohrádky a trilobot nemá dostatek vzorků pro napojení cíle.

7.1.4 Test auta s přívěsem na scéně „Maze“

Poslední test je převážně zaměřen na ověření hledání cest pro vozidla s přívěsem. Rozsáhlá scéna „Maze“ je bludištěm, kterým robot musí najít cestu Na obrázku je robot typu auto s přívěsem 7.3 ve scéně „Maze“, poté co nalezl cestu pomocí algoritmu RRT. Postupně byly testovány všechny algoritmy s různými parametry a různými vozidly.

Ukázalo se, že pro tento problém je nejvhodnější použití algoritmu RRT nebo SRT s menšími počty stromů. Spojování náhodných konfigurací s přívěsem pomocí křivek je již náročnější úkol a prodlužuje tak dobu výpočtu. Jednodušší expanze jsou prováděny řádově rychleji. Pro algoritmus PRM, který spojuje čistě náhodné konfigurace, by byl lepší přístup zvolení dvojrozměrné konfigurace pouze pro pozici s následnou transformací na neholonomní cestu.



Obrázek 7.3: Vozidlo s přívěsem ve scéně „Maze“, vygenerované stromy RRT algoritmem a nalezená cesta (bez optimalizace)

Kapitola 8

Závěr

Cílem této diplomové práce bylo nastudování problematiky hledání cest pomocí pravděpodobnostních algoritmů a zaměření na problematiku hledání cest pro neholonomní objekty. Dalším cílem bylo implementovat a navrhnout ukázkový applet a webové stránky s teoretickým popisem problému a spustitelným appletem. Applet musí umožňovat výběr několika pravděpodobnostních algoritmů a nastavení jejich parametrů, dále editovat tvar prostředí a robota. Applet by měl simulovat zvolený algoritmus na daném prostředí pro zadaného robota.

Pro splnění cílů bylo zapotřebí nalezení zdrojů, které se touto problematikou zabývají, používané technologie prostudovat a aplikovat na problém. Dále navrhnout demonstrační applet a webové stránky s teoretickým popisem. V práci jsou popsány různé druhy neholonomních objektů, několik pravděpodobnostních algoritmů, které je možné použít pro hledání cest, a používané techniky pro neholonomní objekty.

Pro demonstrační applet byly zvoleny jako neholonomní objekty vozidla Ackermanova typu, vozidla s diferenciálním podvozkem a vozidla s přívěsem. Implementovány byly algoritmy PRM, RRT, EST a algoritmus SRT ve dvou variantách (RRT nebo EST expanze stromů). Jako spojovací funkce mezi konfiguracemi neholonomních vozidel bylo použito generování Reeds – Shepp křivek. Applet umožňuje změny prostředí, tvaru a vlastností robota, krokování, zrychlování a zpomalování běhu simulace výpočtu.

Applet byl implementován podle návrhu a umístěn na webových stránkách. Dostupný je na `http://www.stud.fit.vutbr.cz/ xohnhe00/propabilisticplanning` i s popisem práce.

8.1 Zhodnocení testování

Experimentálně byla ověřena funkce appletu a postupně pak všech implementovaných pravděpodobnostních algoritmů. Průběžným testováním bylo nalezeno a následně opraveno několik chyb, které narušovalo funkčnost appletu.

Pro účely testování vzniklo několik scén a druhů robotů (mezi nimi jsou diferenciální roboti, roboti s Ackermanovým řízením a roboti s přívěsem), které prověřily všechny implementované algoritmy v různých situacích. Na těchto scénách byla zkoumána úspěšnost nalezení cesty algoritmů s různými parametry a výsledky byly zaznamenány. Na jejich základě byly algoritmy vzájemně porovnány.

Z výsledků vyplynulo, že algoritmus PRM s rovnoměrným rozložením pravděpodobnostní funkce je vhodný pro rovnoměrně rozložené překážky a má problém dostat se do

kritických úzkých míst.

Dále algoritmus EST je více efektivním v menších uzavřených prostorech, kde má malou možnost šíření do stran. Pravděpodobnostní funkce na bázi hustoty pak tlačí expandované uzly do méně směrů.

Algoritmus RRT zvládá jak dlouhé cesty, tak i malé kroky. Expanduje rychle a rovnoměrně.

Nakonec SRT tyto algoritmy kombinuje do jediného, zadáním určitých parametrů lze získat jak přímo PRM tak i EST nebo RRT. Jako kombinace je velmi efektivní pro řešení všech testovaných problémů.

8.2 Možná rozšíření projektu

Výsledný applet může být rozšířen mnoha způsoby. Byly navrženy následující možnosti rozšíření:

- Přidání nových fyzikálních modelů do appletu, například letadlo (které dovolí pouze cestu jedním směrem, může brát v úvahu i směr a sílu větru), nebo robotické rameno (s velkým počtem kloubů roste počet proměnných v konfiguraci).
- Přidání nových algoritmů, které nebyly implementovány, například SBL. Ale mohou být přidány i jiné než pravděpodobnostní algoritmy hledání cesty.
- Vylepšení optimalizace výsledné cesty, tak aby bral v úvahu i uraženou vzdálenost a dokázal i posouvat existující uzly.
- Přidání možnosti ukládání nových uživatelských scén a vozidel na server.
- Vylepšení funkcí hustoty v EST.
- Paralelizace výpočtu algoritmu SRT.

Literatura

[1] robotika.cz [online]. Dostupné z http://robotika.cz/guide/cs, 2005 [cit. 2012-11-18].

[2] Java SE Technical Documentation [online]. Dostupné z http://docs.oracle.com/javase, [cit. 2013-3-24].

[3] Choset, H.; Lynch, K.; Hutchinson, S.; aj.: *Principles of Robot Motion-Theory, Algorithms, and Implementation*. Cambridge University Press, 2005, ISBN 9780262033275.

[4] Ericson, C.: *Real-Time Collision Detection*. Elsevier, 2005, ISBN 9781558607323.

[5] Herout, P.: *Učebnice jazyka Java*. Kopp, 2000, ISBN 80-7232-115-3.

[6] Šindelář, J.: *Plánování cesty neholonomního mobilního robotu*. Diplomová práce, VUT, 2010.

[7] Krček, P.: *Plánování cesty autonomního lokomočního robotu na základě strojového učení*. Dizertační práce, VUT.

[8] Kvasnica, M.: *Vizualizace hledání cesty pro robota*. Bakalářská práce, VUT, 2008.

[9] LaValle, S. M.: *Planning Algorithms*. Cambridge University Press, 2006, ISBN 9780521862059.

[10] LaValle, S. M.; Kuffner, J. J.: *Rapidly-Exploring RandomTrees: Progress and Prospects*. 2001.

[11] Šíma, M.: *Plánování cesty robota ve spojitém prostředí*. Diplomová práce, VUT, 2006.

[12] Orság, F.: *Robotika*. Studijní opora, VUT, 2006.

[13] Sedlák, V.: *Plánování cesty mobilního robotu pomocí mravenčích algoritmů*. Diplomová práce, VUT, 2011.

[14] Vozak, P.: *Vizualizace plánování cesty*. Bakalářská práce, VUT, 2009.

[15] Zbořil, F.: *Agentní a multiagentní systémy*. Studijní opora, VUT, 2006.

Příloha A

Obsah CD

Přiložený disk CD obsahuje:

- Technická zpráva
- Zdrojový kód technické zprávy ve formátu LaTeX
- Webové stránky projektu
- Zdrojové kódy java appletu
- Spustitelný applet